

Dynamic Template Tracking and Recognition

Rizwan Chaudhry · Gregory Hager · René Vidal

Abstract In this paper we address the problem of tracking non-rigid objects whose local appearance and motion changes as a function of time. This class of objects includes dynamic textures such as steam, fire, smoke, water, etc., as well as articulated objects such as humans performing various actions. We model the temporal evolution of the object's appearance/motion using a Linear Dynamical System (LDS). We learn such models from sample videos and use them as dynamic templates for tracking objects in novel videos. We pose the problem of tracking a dynamic non-rigid object in the current frame as a maximum a-posteriori estimate of the location of the object and the latent state of the dynamical system, given the current image features and the best estimate of the state in the previous frame. The advantage of our approach is that we can specify a-priori the type of texture to be tracked in the scene by using previously trained models for the dynamics of these textures. Our framework naturally generalizes common tracking methods such as SSD and kernel-based tracking from static templates to dynamic templates. We test our algorithm on synthetic as well as real examples of dynamic tex-

tures and show that our simple dynamics-based trackers perform at par if not better than the state-of-the-art. Since our approach is general and applicable to any image feature, we also apply it to the problem of human action tracking and build action-specific optical flow trackers that perform better than the state-of-the-art when tracking a human performing a particular action. Finally, since our approach is generative, we can use a-priori trained trackers for different texture or action classes to simultaneously track and recognize the texture or action in the video.

Keywords Dynamic Templates · Dynamic Textures · Human Actions · Tracking · Linear Dynamical Systems · Recognition

1 Introduction

Object tracking is arguably one of the most important and actively researched areas in computer vision. Accurate object tracking is generally a pre-requisite for vision-based control, surveillance and object recognition in videos. Some of the challenges to accurate object tracking are moving cameras, changing pose, scale and velocity of the object, occlusions, non-rigidity of the object shape and changes in appearance due to ambient conditions. A very large number of techniques have been proposed over the last few decades, each trying to address one or more of these challenges under different assumptions. The comprehensive survey by Yilmaz et al. (2006) provides an analysis of over 200 publications in the general area of object tracking.

In this paper, we focus on tracking objects that undergo non-rigid transformations in shape and appearance as they move around in a scene. Examples of such objects include fire, smoke, water, and fluttering flags, as well as humans performing different actions. Collectively

R. Chaudhry
Center for Imaging Science
Johns Hopkins University
Baltimore, MD
Tel.: +1-410-516-4095
Fax: +1-410-516-4594
E-mail: rizwanch@cis.jhu.edu

G. Hager
Johns Hopkins University
Baltimore, MD
E-mail: hager@cs.jhu.edu

R. Vidal
Center for Imaging Science
Johns Hopkins University
Baltimore, MD
E-mail: rvidal@cis.jhu.edu

called dynamic templates, these objects are fairly common in natural videos. Due to their constantly evolving appearance, they pose a challenge to state-of-the-art tracking techniques that assume consistency of appearance distributions or consistency of shape and contours. However, the change in appearance and motion profiles of dynamic templates is not entirely arbitrary and can be explicitly modeled using Linear Dynamical Systems (LDS). Standard tracking methods either use subspace models or simple Gaussian models to describe appearance changes of a mean template. Other methods use higher-level features such as skeletal structures or contour deformations for tracking to reduce dependence on appearance features. Yet others make use of foreground-background classifiers and learn discriminative features for the purpose of tracking. However, all these methods ignore the temporal dynamics of the appearance changes that are characteristic to the dynamic template.

Over the years, several methods have been developed for segmentation and recognition of dynamic templates, in particular dynamic textures. However, to the best of our knowledge the only work that explicitly addresses tracking of dynamic textures was done by Péteri (2010). As we will describe in detail later, this work also does not consider the temporal dynamics of the appearance changes and does not perform well in experiments.

Paper Contributions and Outline. In the proposed approach, we model the temporal evolution of the appearance of dynamic templates using Linear Dynamical Systems (LDSs) whose parameters are learned from sample videos. These LDSs will be incorporated in a kernel based tracking framework that will allow us to track non-rigid objects in novel video sequences. In the remaining part of this section, we will review some of the related works in tracking and motivate the need for dynamic template tracking method. We will then review static template tracking in §2. In §3, we pose the tracking problem as the maximum a-posteriori estimate of the location of the template as well as the internal state of the LDS, given a kernel-weighted histogram observed at a test location in the image and the internal state of the LDS at the previous frame. This results in a novel joint optimization approach that allows us to simultaneously compute the best location as well as the internal state of the moving dynamic texture at the current time instant in the video. We then show how our proposed approach can be used to perform simultaneous tracking and recognition in §4. In §5, we first evaluate the convergence properties of our algorithm on synthetic data before validating it with experimental results on real datasets of Dynamic Textures and Human Activities in §6, §7 and §8. Finally, we will mention future research directions and conclude in §9.

Prior Work on Tracking Non-Rigid and Articulated Objects. In the general area of tracking, Isard and Blake (1998) and North et al. (2000) hand craft models for object contours using splines and learn their dynamics using Expectation Maximization (EM). They then use particle filtering and Markov Chain Monte-Carlo methods to track and classify the object motion. However for most of the cases, the object contours do not vary significantly during the tracking task. In the case of dynamic textures, generally there is no well-defined contour and hence this approach is not directly applicable. Black and Jepson (1998) propose using a robust appearance subspace model for a known object to track it later in a novel video. However there are no dynamics associated to the appearance changes and in each frame, the projection coefficients are computed independently from previous frames. Jepson et al. (2001) propose an EM-based method to estimate parameters of a mixture model that combines stable object appearance, frame-to-frame variations, and an outlier model for robustly tracking objects that undergo appearance changes. Although, the motivation behind such a model is compelling, its actual application requires heuristics and a large number of parameters. Moreover, dynamic textures do not have a stable object appearance model, instead the appearance changes according to a distinct Gauss-Markov process characteristic to the class of the dynamic texture.

Tracking of non-rigid objects is often motivated by the application of human tracking in videos. In Pavlovic et al. (1999), a Dynamic Bayesian Network is used to learn the dynamics of human motion in a scene. Joint angle dynamics are modeled using switched linear dynamical systems and used for classification, tracking and synthesis of human motion. Although, the tracking results for human skeletons are impressive, extreme care is required to learn the joint dynamic models from manually extracted skeletons or motion capture data. Moreover a separate dynamical system is learnt for each joint angle instead of a global model for the entire object. Approaches such as Leibe et al. (2008) maintain multiple hypotheses for object tracks and continuously refine them as the video progresses using a Minimum Description Length (MDL) framework. The work by Lim et al. (2006) models dynamic appearance by using non-linear dimensionality reduction techniques and learns the temporal dynamics of these low-dimensional representation to predict future motion trajectories. Nejhum et al. (2008) propose an online approach that deals with appearance changes due to articulation by updating the foreground shape using rectangular blocks that adapt to find the best match in every frame. However foreground

appearance is assumed to be stationary throughout the video.

Recently, classification based approaches have been proposed in Grabner et al. (2006) and Babenko et al. (2009) where classifiers such as boosting or Multiple Instance Learning are used to adapt foreground vs background appearance models with time. This makes the tracker invariant to gradual appearance changes due to object rotation, illumination changes etc. This discriminative approach, however, does not incorporate an inherent temporal model of appearance variations, which is characteristic of, and potentially very useful, for dynamic textures.

In summary, all the above works lack a unified framework that simultaneously models the temporal dynamics of the object appearance and shape as well as the motion through the scene. Moreover, most of the works concentrate on handling the appearance changes due to articulation and are not directly relevant to dynamic textures where there is no articulation.

Prior Work on Tracking Dynamic Templates. Recently, Péteri (2010) propose a first method for tracking dynamic textures using a particle filtering approach similar to the one presented in Isard and Blake (1998). However their approach can best be described as a static template tracker that uses optical flow features. The method extracts histograms for the magnitude, direction, divergence and curl of the optical flow field of the dynamic texture in the first two frames. It then assumes that the change in these flow characteristics with time can simply be modeled using a Gaussian distribution with the initially computed histograms as the mean. The variance of this distribution is selected as a parameter. Furthermore, they do not model the characteristic temporal dynamics of the intensity variations specific to each class of dynamic textures, most commonly modeled using LDSs. As we will also show in our experiments, their approach performs poorly on several real dynamic texture examples.

LDS-based techniques have been shown to be extremely valuable for dynamic texture recognition (Saisan et al., 2001; Doretto et al., 2003; Chan and Vasconcelos, 2007; Ravichandran et al., 2009), synthesis (Doretto et al., 2003), and registration (Ravichandran and Vidal, 2008). They have also been successfully used to model the temporal evolution of human actions for the purpose of activity recognition (Bissacco et al., 2001, 2007; Chaudhry et al., 2009). Therefore, it is only natural to assume that such a representation should also be useful for tracking.

Finally, Vidal and Ravichandran (2005) propose a method to jointly compute the dynamics as well as the optical flow of a scene for the purpose of segmenting

moving dynamic textures. Using the Dynamic Texture Constancy Constraint (DTCC), the authors show that if the motion of the texture is slow, the optical flow corresponding to 2-D rigid motion of the texture (or equivalently the motion of the camera) can be computed using a method similar to the Lucas-Kanade optical flow algorithm. In principle, this method can be extended to track a dynamic texture in a framework similar to the KLT tracker. However, the requirement of having a slow-moving dynamic texture is particularly strict, especially for high-order systems and would not work in most cases. Moreover, the authors do not enforce any spatial coherence of the moving textures, which causes the segmentation results to have holes.

In light of the above discussion, we posit that there is a need to develop a principled approach for tracking dynamic templates that explicitly models the characteristic temporal dynamics of the appearance and motion. As we will show, by incorporating these dynamics, our proposed method achieves superior tracking results as well as allows us to perform simultaneous tracking and recognition of dynamic templates.

2 Review of Static Template Tracking

In this section, we will formulate the tracking problem as a maximum a-posteriori estimation problem and show how standard static template tracking methods such as Sum-of-Squared-Differences (SSD) and kernel-based tracking are special cases of this general problem.

Assume that we are given a static template $\mathcal{I} : \Omega \rightarrow \mathbb{R}$, centered at the origin on the discrete pixel grid, $\Omega \subset \mathbb{R}^2$. At each time instant, t , we observe an image frame, $\mathbf{y}_t : \mathcal{F} \rightarrow \mathbb{R}$, where $\mathcal{F} \subset \mathbb{R}^2$ is the discrete pixel domain of the image. As the template moves in the scene, it undergoes a translation, $\mathbf{l}_t \in \mathbb{R}^2$, from the origin of the template reference frame Ω . Moreover, assume that due to noise in the observed image the intensity at each pixel in \mathcal{F} is corrupted by i.i.d. Gaussian noise with mean 0, and standard deviation σ_Y . Hence, for each pixel location $\mathbf{z} \in \Omega + \mathbf{l}_t = \{\mathbf{z}' + \mathbf{l}_t : \mathbf{z}' \in \Omega\}$, we have,

$$\mathbf{y}_t(\mathbf{z}) = \mathcal{I}(\mathbf{z} - \mathbf{l}_t) + \mathbf{w}_t(\mathbf{z}), \text{ where } \mathbf{w}_t(\mathbf{z}) \stackrel{\text{iid}}{\sim} \mathcal{N}(0, \sigma_Y^2). \quad (1)$$

Therefore, the likelihood of the image intensity at pixel $\mathbf{z} \in \Omega + \mathbf{l}_t$ is $p(\mathbf{y}_t(\mathbf{z})|\mathbf{l}_t) = \mathcal{G}_{\mathbf{y}_t(\mathbf{z})}(\mathcal{I}(\mathbf{z} - \mathbf{l}_t), \sigma_Y^2)$, where

$$\mathcal{G}_{\mathbf{x}}(\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \mu)^\top \Sigma^{-1} (\mathbf{x} - \mu) \right\}$$

is the n -dimensional Gaussian pdf with mean μ and covariance Σ . Given $\mathbf{y}_t = [\mathbf{y}_t(\mathbf{z})]_{\mathbf{z} \in \mathcal{F}}$, i.e., the stack of all the pixel intensities in the frame at time t , we would like to maximize the posterior, $p(\mathbf{l}_t|\mathbf{y}_t)$. Assuming a

uniform background, i.e., $p(\mathbf{y}_t(\mathbf{z})|\mathbf{l}_t) = 1/K$ if $\mathbf{z} - \mathbf{l}_t \notin \Omega$ and a uniform prior for \mathbf{l}_t , i.e., $p(\mathbf{l}_t) = |\mathcal{F}|^{-1}$, we have,

$$\begin{aligned} p(\mathbf{l}_t|\mathbf{y}_t) &= \frac{p(\mathbf{y}_t|\mathbf{l}_t)p(\mathbf{l}_t)}{p(\mathbf{y}_t)} \\ &\propto p(\mathbf{y}_t|\mathbf{l}_t) = \prod_{\mathbf{z} \in \Omega + \mathbf{l}_t} p(\mathbf{y}_t(\mathbf{z})|\mathbf{l}_t) \\ &\propto \exp \left\{ -\frac{1}{2\sigma_Y^2} \sum_{\mathbf{z} \in \Omega + \mathbf{l}_t} (\mathbf{y}_t(\mathbf{z}) - \mathcal{I}(\mathbf{z} - \mathbf{l}_t))^2 \right\}. \end{aligned} \quad (2)$$

The optimum value, $\hat{\mathbf{l}}_t$ will maximize the log posterior and after some algebraic manipulations, we get

$$\hat{\mathbf{l}}_t = \underset{\mathbf{l}_t}{\operatorname{argmin}} \sum_{\mathbf{z} \in \Omega + \mathbf{l}_t} (\mathbf{y}_t(\mathbf{z}) - \mathcal{I}(\mathbf{z} - \mathbf{l}_t))^2. \quad (3)$$

Notice that with the change of variable, $\mathbf{z}' = \mathbf{z} - \mathbf{l}_t$, we can shift the domain from the image pixel space \mathcal{F} to the template pixel space Ω , to get $\hat{\mathbf{l}}_t = \underset{\mathbf{l}_t}{\operatorname{argmin}} O(\mathbf{l}_t)$, where

$$O(\mathbf{l}_t) = \sum_{\mathbf{z}' \in \Omega} (\mathbf{y}_t(\mathbf{z}' + \mathbf{l}_t) - \mathcal{I}(\mathbf{z}'))^2. \quad (4)$$

Eq. (4) is the well known optimization function used in Sum of Squared Differences (SSD)-based tracking of static templates. The optimal solution is found either by searching brute force over the entire image frame or, given an initial estimate of the location, by performing gradient descent iterations: $\mathbf{l}_t^{i+1} = \mathbf{l}_t^i - \gamma \nabla_{\mathbf{l}_t} O(\mathbf{l}_t^i)$. Since the image intensity function, \mathbf{y}_t is non-convex, a good initial guess, \mathbf{l}_t^0 , is very important for the gradient descent algorithm to converge to the correct location. Generally, \mathbf{l}_t^0 is initialized using the optimal location found at the previous time instant, i.e., $\hat{\mathbf{l}}_{t-1}$, and $\hat{\mathbf{l}}_0$ is hand set or found using an object detector.

In the above description, we have assumed that we observe the intensity values, \mathbf{y}_t , directly. However, to develop an algorithm that is invariant to nuisance factors such as changes in contrast and brightness, object orientations, etc., we can choose to compute the value of a more robust function that also considers the intensity values over a neighborhood $\Gamma(\mathbf{z}) \subset \mathbb{R}^2$ of the pixel location \mathbf{z} ,

$$\mathbf{f}_t(\mathbf{z}) = f([\mathbf{y}_t(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z})}), \quad f: \mathbb{R}^{|\Gamma|} \rightarrow \mathbb{R}^d, \quad (5)$$

where $[\mathbf{y}_t(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z})}$ represents the stack of intensity values of all the pixel locations in $\Gamma(\mathbf{z})$. We can therefore treat the value of $\mathbf{f}_t(\mathbf{z})$ as the observed random variable at the location \mathbf{z} instead of the actual intensities, $\mathbf{y}_t(\mathbf{z})$.

Notice that even though the conditional probability of the intensity of individual pixels is Gaussian, as in Eq. (1), under the (possibly) non-linear transformation, f , the conditional probability of $\mathbf{f}_t(\mathbf{z})$ will no longer be

Gaussian. However, from an empirical point of view, using a Gaussian assumption in general provides very good results. Therefore, due to changes in the location of the template, we observe $\mathbf{f}_t(\mathbf{z}) = f([\mathcal{I}(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z} - \mathbf{l}_t)}) + \mathbf{w}_t^f(\mathbf{z})$, where $\mathbf{w}_t^f(\mathbf{z}) \stackrel{\text{iid}}{\sim} \mathcal{N}(0, \sigma_f^2)$ is isotropic Gaussian noise.

Following the same derivation as before, the new cost function to be optimized becomes,

$$\begin{aligned} O(\mathbf{l}_t) &= \sum_{\mathbf{z} \in \Omega} \|f([\mathbf{y}_t(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z} + \mathbf{l}_t)}) - f([\mathcal{I}(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z})})\|^2 \\ &\doteq \|F(\mathbf{y}_t(\mathbf{l}_t)) - F(\mathcal{I})\|^2, \end{aligned} \quad (6)$$

where,

$$\begin{aligned} F(\mathbf{y}_t(\mathbf{l}_t)) &\doteq [f([\mathbf{y}_t(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z} + \mathbf{l}_t)})]_{\mathbf{z} \in \Omega} \\ F(\mathcal{I}) &\doteq [f([\mathcal{I}(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z})})]_{\mathbf{z} \in \Omega} \end{aligned}$$

By the same argument as in Eq. (4), $\hat{\mathbf{l}}_t = \underset{\mathbf{l}_t}{\operatorname{argmin}} O(\mathbf{l}_t)$ also maximizes the posterior, $p(\mathbf{l}_t|F(\mathbf{y}_t))$, where

$$F(\mathbf{y}_t) \doteq [f([\mathbf{y}_t(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z})})]_{\mathbf{z} \in \mathcal{F}}, \quad (7)$$

is the stack of all the function evaluations with neighborhood size Γ over all the pixels in the frame.

For the sake of simplicity, from now on as in Eq. (6), we will abuse the notation and use $\mathbf{y}_t(\mathbf{l}_t)$ to denote the stacked vector $[\mathbf{y}_t(\mathbf{z}')]_{\mathbf{z}' \in \Gamma(\mathbf{z} + \mathbf{l}_t)}$, and \mathbf{y}_t to denote the full frame, $[\mathbf{y}_t(\mathbf{z})]_{\mathbf{z} \in \mathcal{F}}$. Moreover, assume that the ordering of pixels in Ω is in column-wise format, denoted by the set $\{1, \dots, N\}$. Finally, if the size of the neighborhood, Γ , is equal to the size of the template, \mathcal{I} , i.e., $|\Gamma| = |\Omega|$, f will only need to be computed at the central pixel of Ω , shifted by \mathbf{l}_t , i.e.,

$$O(\mathbf{l}_t) = \|f(\mathbf{y}_t(\mathbf{l}_t)) - f(\mathcal{I})\|^2 \quad (8)$$

Kernel based Tracking. One special class of functions that has commonly been used in *kernel-based tracking* methods (Comaniciu and Meer, 2002; Comaniciu et al., 2003) is that of kernel-weighted histograms of intensities. These functions have very useful properties in that, with the right choice of the kernel, they can be made either robust to variations in the pose of the object, or sensitive to certain discriminatory characteristics of the object. This property is extremely useful in common tracking problems and is the reason for the wide use of kernel-based tracking methods. In particular, a kernel-weighted histogram, $\rho = [\rho_1, \dots, \rho_B]^\top$ with B bins, $u = 1, \dots, B$, computed at pixel location \mathbf{l}_t , is defined as,

$$\rho_u(\mathbf{y}_t(\mathbf{l}_t)) = \frac{1}{\kappa} \sum_{\mathbf{z} \in \Omega} K(\mathbf{z}) \delta(b(\mathbf{y}_t(\mathbf{z} + \mathbf{l}_t)) - u), \quad (9)$$

where b is a binning function for the intensity $\mathbf{y}_t(\mathbf{z})$ at the pixel \mathbf{z} , δ is the discrete Kronecker delta function, and $\kappa = \sum_{\mathbf{z} \in \Omega} K(\mathbf{z})$ is a normalization constant such that the sum of the histogram equals 1. One of the more commonly used kernels is the Epanechnikov kernel,

$$K(\mathbf{z}) = \begin{cases} 1 - \|H\mathbf{z}\|^2, & \|H\mathbf{z}\| < 1 \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

where $H = \text{diag}([r^{-1}, c^{-1}])$ is the bandwidth scaling matrix of the kernel corresponding to the size of the template, i.e., $|\Omega| = r \times c$.

Using the fact that we observe $F = \sqrt{\rho}$, the entry-wise square root of the histogram, we get the Matusita metric between the kernel weighted histogram computed at the current location in the image frame and that of the template:

$$O(\mathbf{l}_t) = \|\sqrt{\rho(\mathbf{y}_t(\mathbf{l}_t))} - \sqrt{\rho(\mathcal{I})}\|^2. \quad (11)$$

Hager et al. (2004) showed that the minimizer of the objective function in Eq. (11) is precisely the solution of the meanshift tracker as originally proposed by Comaniciu and Meer (2002) and Comaniciu et al. (2003). The algorithm in Hager et al. (2004) then proceeds by computing the optimal \mathbf{l}_t that minimizes Eq. (11) using a Newton-like approach. We refer the reader to Hager et al. (2004) for more details. Hager et al. (2004) then propose using multiple kernels and Fan et al. (2007) propose structural constraints to get unique solutions in difficult cases. All these formulations eventually boil down to the solution of a problem of the form in Eq. (8).

Incorporating Location Dynamics. The generative model in Eq. (2) assumes a uniform prior on the probability of the location of the template at time t and that the location at time t is independent of the location at time $t - 1$. If applicable, we can improve the performance of the tracker by imposing a known motion model, $\mathbf{l}_t = g(\mathbf{l}_{t-1}) + \mathbf{w}_t^g$, such as constant velocity or constant acceleration. In this case, the likelihood model is commonly appended by,

$$p(\mathbf{l}_t | \mathbf{l}_{t-1}) = \mathcal{G}_{\mathbf{l}_t}(g(\mathbf{l}_{t-1}), \Sigma_g). \quad (12)$$

From here, it is a simple exercise to see that the maximum a-posteriori estimate of \mathbf{l}_t given all the frames, $\mathbf{y}_0, \dots, \mathbf{y}_t$ can be computed by the extended Kalman filter or particle filters since f , in Eq. (8), is a function of the image intensities and therefore a non-linear function on the pixel domain.

3 Tracking Dynamic Templates

In the previous section we reviewed kernel-based methods for tracking a static template $\mathcal{I} : \Omega \rightarrow \mathbb{R}$. In this section we propose a novel kernel-based framework for tracking a dynamic template $\mathcal{I}_t : \Omega \rightarrow \mathbb{R}$. For ease of exposition, we derive the framework under the assumption that the location of the template \mathbf{l}_t is equally likely on the image domain. For the case of a dynamic prior on the location, the formulation will result in an extended Kalman or particle filter as briefly mentioned at the end of §2.

We model the temporal evolution of the dynamic template \mathcal{I}_t using Linear Dynamical Systems (LDSs). LDSs are represented by the tuple (μ, A, C, B) and satisfy the following equations for all time t :

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + B\mathbf{v}_t, \quad (13)$$

$$\mathcal{I}_t = \mu + C\mathbf{x}_t. \quad (14)$$

Here $\mathcal{I}_t \in \mathbb{R}^{|\Omega|}$ is the stacked vector, $[\mathcal{I}_t(\mathbf{z})]_{\mathbf{z} \in \Omega}$, of image intensities of the dynamic template at time t , and \mathbf{x}_t is the (hidden) state of the system at time t . The current state is linearly related to the previous state by the state transition matrix A and the current output is linearly related to the current state by the observation matrix C . \mathbf{v}_t is the process noise, which is assumed to be Gaussian and independent from \mathbf{x}_t . Specifically, $B\mathbf{v}_t \sim \mathcal{N}(0, Q)$, where $Q = BB^\top$.

Tracking dynamic templates requires knowledge of the system parameters, (μ, A, C, B) , for dynamic templates of interest. Naturally, these parameters have to be learnt from training data. Once these parameters have been learnt, they can be used to track the template in a new video. However the size, orientation, and direction of motion of the template in the test video might be very different from that of the training videos and therefore our procedure will need to be invariant to these changes. In the following, we will propose our dynamic template tracking framework by describing in detail each of these steps,

1. Learning the system parameters of a dynamic template from training data,
2. Tracking dynamic templates of the same size, orientation, and direction of motion as training data,
3. Discussing the convergence properties and parameter tuning, and
4. Incorporating invariance to size, orientation, and direction of motion.

3.1 LDS Parameter Estimation

We will first describe the procedure to learn the system parameters, (μ, A, C, B) , of a dynamic template from a

training video of that template. Assuming that we can manually, or semi-automatically, mark a bounding box of size $|\Omega| = r \times c$ rows and columns, which best covers the spatial extent of the dynamic template at each frame of the training video. The center of each box gives us the location of the template at each frame, which we use as the ground truth template location. Next, we extract the sequence of column-wise stacked pixel intensities, $\mathcal{I} = [\mathcal{I}_1, \dots, \mathcal{I}_N]$ corresponding to the appearance of the template at each time instant in the marked bounding box. We can then compute the system parameters using the system identification approach proposed in [Doretto et al. \(2003\)](#). Briefly, given the sequence, \mathcal{I} , we compute a compact, rank n , singular value decomposition (SVD) of the matrix, $\tilde{\mathcal{I}} = [\mathcal{I}_1 - \mu, \dots, \mathcal{I}_N - \mu] = U \Sigma V^\top$. Here $\mu = \frac{1}{N} \sum_{i=1}^N \mathcal{I}_i$, and n is the order of the system and is a parameter. For all our experiments, we have chosen $n = 5$. We then compute $C = U$, and the state sequence $X_1^N = \Sigma V^\top$, where $X_{t_1}^{t_2} = [\mathbf{x}_{t_1}, \dots, \mathbf{x}_{t_2}]$. Given X_1^N , the matrix A can be computed using least-squares as $A = X_2^N (X_1^{N-1})^\dagger$, where X^\dagger represents the pseudo-inverse of X . Also, $Q = \frac{1}{N-1} \sum_{t=1}^{N-1} \mathbf{v}'_t (\mathbf{v}'_t)^\top$ where $\mathbf{v}'_t = B \mathbf{v}_t = \mathbf{x}_{t+1} - \mathbf{x}_t$. B is computed using the Cholesky factorization of $Q = BB^\top$.

3.2 Tracking Dynamic Templates

Problem Formulation. We will now formulate the problem of tracking a dynamic template of size $|\Omega| = r \times c$, with known system parameters (μ, A, C, B) . Given a test video, at each time instant, t , we observe an image frame, $\mathbf{y}_t : \mathcal{F} \rightarrow \mathbb{R}$, obtained by translating the template \mathcal{I}_t by an amount $\mathbf{l}_t \in \mathbb{R}^2$ from the origin of the template reference frame Ω . Previously, at time instant $t - 1$, the template was observed at location \mathbf{l}_{t-1} in frame \mathbf{y}_{t-1} . In addition to the change in location, the intensity of the dynamic template changes according to Eqs. (13-14). Moreover, assume that due to noise in the observed image the intensity at each pixel in \mathcal{F} is corrupted by i.i.d. Gaussian noise with mean 0, and standard deviation σ_Y . Therefore, the intensity at pixel $\mathbf{z} \in \mathcal{F}$ given the location of the template and the current state of the dynamic texture is

$$\mathbf{y}_t(\mathbf{z}) = \mathcal{I}_t(\mathbf{z} - \mathbf{l}_t) + \mathbf{w}_t(\mathbf{z}) \quad (15)$$

$$= \mu(\mathbf{z} - \mathbf{l}_t) + C(\mathbf{z} - \mathbf{l}_t)^\top \mathbf{x}_t + \mathbf{w}_t(\mathbf{z}), \quad (16)$$

where the pixel \mathbf{z} is used to index μ in Eq. (13) according to the ordering in Ω , e.g., in a column-wise fashion. Similarly, $C(\mathbf{z})^\top$ is the row of the C matrix in Eq. (13) indexed by the pixel \mathbf{z} . Fig. 1 illustrates the tracking scenario and Fig. 2 shows the corresponding graphical

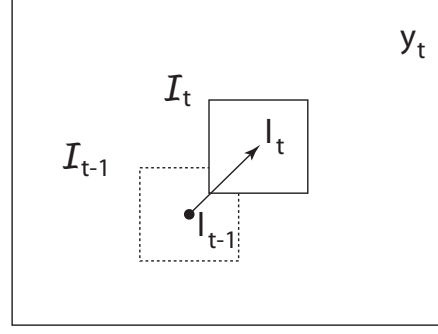


Fig. 1 Illustration of the dynamic template tracking problem.

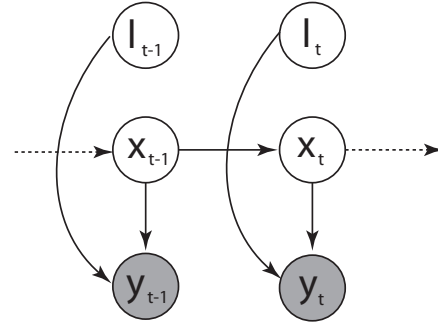


Fig. 2 Graphical representation for the generative model of the observed template.

model representation¹. We only observe the frame, \mathbf{y}_t and the appearance of the frame is conditional on the location of the dynamic template, \mathbf{l}_t and its state, \mathbf{x}_t .

As described in §2, rather than using the image intensities \mathbf{y}_t as our measurements, we compute a kernel-weighted histogram centered at each test location \mathbf{l}_t ,

$$\rho_u(\mathbf{y}_t(\mathbf{l}_t)) = \frac{1}{\kappa} \sum_{\mathbf{z} \in \Omega} K(\mathbf{z}) \delta(b(\mathbf{y}_t(\mathbf{z} + \mathbf{l}_t)) - u). \quad (17)$$

In an entirely analogous fashion, we compute a kernel-weighted histogram of the template

$$\rho_u(\mathcal{I}_t(\mathbf{x}_t)) = \frac{1}{\kappa} \sum_{\mathbf{z} \in \Omega} K(\mathbf{z}) \delta(b(\mu(\mathbf{z}) + C(\mathbf{z})^\top \mathbf{x}_t) - u), \quad (18)$$

where we write $\mathcal{I}_t(\mathbf{x}_t)$ to emphasize the dependence of the template \mathcal{I}_t on the latent state \mathbf{x}_t , which needs to be estimated together with the template location \mathbf{l}_t .

Since the space of histograms is a non-Euclidean space, we need a metric on the space of histograms to be able to correctly compare the observed kernel-weighted histogram with that generated by the template. One convenient metric on the space of histograms is the

¹ Notice that since we have assumed that the location of the template at time t is independent of its location at time $t - 1$, there is no link from \mathbf{l}_{t-1} to \mathbf{l}_t in the graphical model.

Matusita metric,

$$d(\rho^1, \rho^2) = \sum_{i=1}^B \left(\sqrt{\rho_i^1} - \sqrt{\rho_i^2} \right)^2 \quad (19)$$

which was also previously used in Eq. (11) for the static template case by Hager et al. (2004). Therefore, we approximate the probability of the square root of each entry of the histogram as,

$$p(\sqrt{\rho_u(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) \approx \mathcal{G}_{\sqrt{\rho_u(\mathbf{y}_t(\mathbf{l}_t))}}(\sqrt{\rho_u(\mathcal{I}_t(\mathbf{x}_t))}, \sigma_H^2), \quad (20)$$

where σ_H^2 is the variance of the entries of the histogram bins. The tracking problem therefore results in the maximum a-posteriori estimation,

$$(\hat{\mathbf{l}}_t, \hat{\mathbf{x}}_t) = \underset{\mathbf{l}_t, \mathbf{x}_t}{\operatorname{argmax}} p(\mathbf{l}_t, \mathbf{x}_t | \sqrt{\rho(\mathbf{y}_1)}, \dots, \sqrt{\rho(\mathbf{y}_t)}) \quad (21)$$

where $\{\rho(\mathbf{y}_i)\}_{i=1}^t$ are the kernel-weighted histograms computed at each image location in all the frames with the square-root taken element-wise. Deriving an optimal non-linear filter for the above problem might not be computationally feasible and we might have to resort to particle filtering based approaches. However, as we will explain, we can simplify the above problem drastically by proposing a greedy solution that, although not provably optimal, is computationally efficient. Moreover, as we will show in the experimental section, this solution results in an algorithm that performs at par or even better than state-of-the-art tracking methods.

Bayesian Filtering. Define $\mathbf{P}_t = \{\sqrt{\rho(\mathbf{y}_1)} \dots \sqrt{\rho(\mathbf{y}_t)}\}$, and consider the Bayesian filter derivation for Eq. (21):

$$\begin{aligned} p(\mathbf{l}_t, \mathbf{x}_t | \mathbf{P}_t) &= p(\mathbf{l}_t, \mathbf{x}_t | \mathbf{P}_{t-1}, \sqrt{\rho(\mathbf{y}_t)}) \\ &= \frac{p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) p(\mathbf{l}_t, \mathbf{x}_t | \mathbf{P}_{t-1})}{p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{P}_{t-1})} \\ &\propto p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) p(\mathbf{l}_t, \mathbf{x}_t | \mathbf{P}_{t-1}) \\ &= p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) \int_{\mathcal{X}_{t-1}} p(\mathbf{l}_t, \mathbf{x}_t, \mathbf{x}_{t-1} | \mathbf{P}_{t-1}) d\mathbf{x}_{t-1} \\ &= p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) \cdot \\ &\quad \int_{\mathcal{X}_{t-1}} p(\mathbf{l}_t, \mathbf{x}_t | \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} | \mathbf{P}_{t-1}) d\mathbf{x}_{t-1} \\ &= p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) p(\mathbf{l}_t). \\ &\quad \int_{\mathcal{X}_{t-1}} p(\mathbf{x}_t | \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} | \mathbf{P}_{t-1}) d\mathbf{x}_{t-1}, \end{aligned}$$

where we have assumed a uniform prior $p(\mathbf{l}_t) = |\mathcal{F}|^{-1}$. Assuming that we have full confidence in the estimate $\hat{\mathbf{x}}_{t-1}$ of the state at time $t-1$, we can use the *greedy*

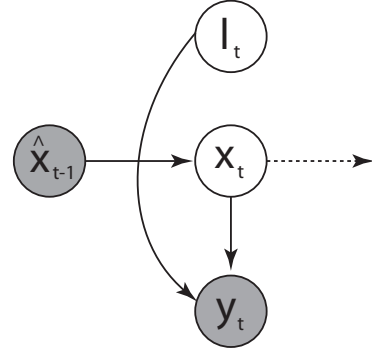


Fig. 3 Graphical Model for the approximate generative model of the observed template

posterior, $p(\mathbf{x}_{t-1} | \mathbf{P}_{t-1}) = \delta(\mathbf{x}_{t-1} = \hat{\mathbf{x}}_{t-1})$, to greatly simplify Eq. (22) as,

$$\begin{aligned} p(\mathbf{l}_t, \mathbf{x}_t | \mathbf{P}_t) &\propto p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{x}_{t-1} = \hat{\mathbf{x}}_{t-1}) p(\mathbf{l}_t) \\ &\propto p(\sqrt{\rho(\mathbf{y}_t)} | \mathbf{l}_t, \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{x}_{t-1} = \hat{\mathbf{x}}_{t-1}). \end{aligned} \quad (22)$$

Fig. 3 shows the graphical model corresponding to this approximation.

After some algebraic manipulations, we arrive at,

$$(\hat{\mathbf{l}}_t, \hat{\mathbf{x}}_t) = \underset{\mathbf{l}_t, \mathbf{x}_t}{\operatorname{argmin}} O(\mathbf{l}_t, \mathbf{x}_t) \quad (23)$$

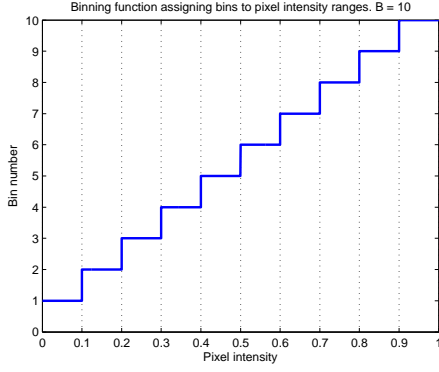
where,

$$\begin{aligned} O(\mathbf{l}_t, \mathbf{x}_t) &= \frac{1}{2\sigma_H^2} \|\sqrt{\rho(\mathbf{y}_t(\mathbf{l}_t))} - \sqrt{\rho(\mu + C\mathbf{x}_t)}\|^2 + \\ &\quad \frac{1}{2} (\mathbf{x}_t - A\hat{\mathbf{x}}_{t-1})^\top Q^{-1} (\mathbf{x}_t - A\hat{\mathbf{x}}_{t-1}). \end{aligned} \quad (24)$$

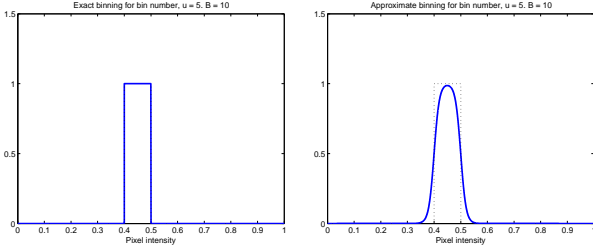
Simultaneous State Estimation and Tracking. To derive a gradient descent scheme, we need to take derivatives of O w.r.t. \mathbf{x}_t . However, the histogram function, ρ is not differentiable w.r.t. \mathbf{x}_t because of the δ function and hence we cannot compute the required derivative of Eq. (24). Instead of ρ , we propose to use ζ , where ζ is a *continuous* histogram function defined as,

$$\begin{aligned} \zeta_u(\mathbf{y}_t(\mathbf{l}_t)) &= \frac{1}{\kappa} \sum_{\mathbf{z} \in \Omega} K(\mathbf{z}) \\ &\quad (\phi_{u-1}(\mathbf{y}_t(\mathbf{z} + \mathbf{l}_t)) - \phi_u(\mathbf{y}_t(\mathbf{z} + \mathbf{l}_t))), \end{aligned} \quad (25)$$

where $\phi_u(s) = (1 + \exp\{-\sigma(s - r(u))\})^{-1}$ is the sigmoid function. With a suitably chosen σ ($\sigma = 100$ in our case), the difference of the two sigmoids is a continuous and differentiable function that approximates a step function on the grayscale intensity range $[r(u-1), r(u)]$. For example, for a histogram with B bins, to uniformly cover the grayscale intensity range from 0 to 1, we have $r(u) = \frac{u}{B}$. The difference, $\phi_{u-1}(y) - \phi_u(y)$, will therefore give a value close to 1 when the pixel intensity is



(a) Binning function, $b(\cdot)$, for a histogram with 10 bins, $B = 10$.



(b) Exact binning

(c) Approximate binning

Fig. 4 Binning function for $B = 10$ bins and the corresponding non-differentiable exact binning strategy vs our proposed differentiable but approximate binning strategy, for bin number $u = 5$.

in the range $[\frac{u-1}{B}, \frac{u}{B}]$, and close to 0 otherwise, thus contributing to the u -th bin. Fig. 4(a) illustrates the binning function $b(\cdot)$ in Eqs. (9, 17, 18) for a specific case where the pixel intensity range $[0, 1]$ is equally divided into $B = 10$ bins. Fig. 4(b) shows the non-differentiable but exact binning function, $\delta(b(\cdot), u)$, for bin number $u = 5$, whereas Fig. 4(c) shows the corresponding proposed approximate but differentiable binning function, $(\phi_{u-1} - \phi_u)(\cdot)$. As we can see, the proposed function responds with a value close to 1 for intensity values between $r(5 - 1) = 0.4$, and $r(5) = 0.5$. The spatial kernel weighting of the values is done in exactly the same way as for the non-continuous case in Eq. (9).

We can now find the optimal location and state at each time-step by performing gradient descent on Eq. (24) with ρ replaced by ζ . This gives us the following iterations (see Appendix A for a detailed derivation)

$$\begin{bmatrix} \mathbf{l}_t^{i+1} \\ \mathbf{x}_t^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{l}_t^i \\ \mathbf{x}_t^i \end{bmatrix} - 2\gamma \begin{bmatrix} \mathbf{L}_t^\top \mathbf{a}_i \\ -\mathbf{M}_i^\top \mathbf{a}_i + \mathbf{d}_i \end{bmatrix}, \quad (26)$$

where,

$$\begin{aligned} \mathbf{a} &= \sqrt{\zeta(\mathbf{y}_t(\mathbf{l}_t))} - \sqrt{\zeta(\mu + C\mathbf{x}_t)} \\ \mathbf{d} &= Q^{-1}(\mathbf{x}_t - A\hat{\mathbf{x}}_{t-1}) \\ \mathbf{L} &= \frac{1}{2\sigma_H^2} \text{diag}(\zeta(\mathbf{y}_t(\mathbf{l}_t)))^{-\frac{1}{2}} \tilde{\mathbf{U}}^\top \mathbf{J}_K \\ \mathbf{M} &= \frac{1}{2\sigma_H^2} \text{diag}(\zeta(\mu + C\mathbf{x}_t))^{-\frac{1}{2}} (\Phi')^\top \frac{1}{\kappa} \text{diag}(\mathbf{K}) C. \end{aligned}$$

The index i in Eq. (26) represents evaluation of the above quantities using the estimates $(\mathbf{l}_t^i, \mathbf{x}_t^i)$ at iteration i . Here, \mathbf{J}_K is the Jacobian of the kernel K ,

$$\mathbf{J}_K = [\nabla K(\mathbf{z}_1) \dots \nabla K(\mathbf{z}_N)],$$

and $\tilde{\mathbf{U}} = [\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2, \dots, \tilde{\mathbf{u}}_B]$ is a real-valued *sifting* matrix (analogous to that in Hager et al. (2004)) with,

$$\tilde{\mathbf{u}}_j = \begin{bmatrix} \phi_{j-1}(\mathbf{y}_t(\mathbf{z}_1)) - \phi_j(\mathbf{y}_t(\mathbf{z}_1)) \\ \phi_{j-1}(\mathbf{y}_t(\mathbf{z}_2)) - \phi_j(\mathbf{y}_t(\mathbf{z}_2)) \\ \vdots \\ \phi_{j-1}(\mathbf{y}_t(\mathbf{z}_N)) - \phi_j(\mathbf{y}_t(\mathbf{z}_N)) \end{bmatrix},$$

where the numbers $1, \dots, N$ provide an index in the pixel domain of Ω , as previously mentioned. $\Phi' = [\Phi'_1, \Phi'_2, \dots, \Phi'_B] \in \mathbb{R}^{N \times B}$ is a matrix composed of derivatives of the difference of successive sigmoid functions with,

$$\Phi'_j = \begin{bmatrix} (\phi'_{j-1} - \phi'_j)(\mu(\mathbf{z}_1) + C(\mathbf{z}_1)^\top \mathbf{x}_t) \\ (\phi'_{j-1} - \phi'_j)(\mu(\mathbf{z}_2) + C(\mathbf{z}_2)^\top \mathbf{x}_t) \\ \vdots \\ (\phi'_{j-1} - \phi'_j)(\mu(\mathbf{z}_N) + C(\mathbf{z}_N)^\top \mathbf{x}_t) \end{bmatrix}. \quad (27)$$

Initialization. Solving Eq. (26) iteratively will simultaneously provide the location of the dynamic texture in the scene as well as the internal state of the dynamical system. However, notice that the function O in Eq. (24) is not convex in the variables \mathbf{x}_t and \mathbf{l}_t , and hence the above iterative solution can potentially converge to local minima. To alleviate this to some extent, it is possible to choose a good initialization of the state and location as $\mathbf{x}_t^0 = A\hat{\mathbf{x}}_{t-1}$, and $\mathbf{l}_t^0 = \hat{\mathbf{l}}_{t-1}$. To initialize the tracker in the first frame, we use \mathbf{l}_0 as the initial location marked by the user, or determined by a detector. To initialize \mathbf{x}_0 , we use the pseudo-inverse computation,

$$\hat{\mathbf{x}}_0 = C^\dagger(\mathbf{y}_0(\mathbf{l}_0) - \mu), \quad (28)$$

which coincides with the maximum a-posteriori estimate of the initial state given the correct initial location and the corresponding texture at that time. A good value of the step-size γ can be chosen using any standard step-size selection procedure Gill et al. (1987) such as the Armijo step-size strategy.

We call the above method in its entirety, Dynamic Kernel SSD Tracking (**DK-SSD-T**). For ease of exposition, we have considered the case of a single kernel, however the derivation for multiple stacked and additive kernels [Hager et al. \(2004\)](#), and multiple collaborative kernels with structural constraints [Fan et al. \(2007\)](#) follows similarly.

3.3 Convergence analysis and parameter selection

Convergence of Location. We would first like to discuss the case when the template is static, and can be represented completely using the mean, μ . This is similar to the case when we can accurately synthesize the expected dynamic texture at a particular time instant before starting the tracker. In this case, our proposed approach is analogous to the original meanshift algorithm ([Comaniciu et al., 2003](#)) and follows all the (local) convergence guarantees for that method.

Convergence of State. The second case, concerning the convergence of the state estimate is more interesting. In traditional filtering approaches such as the Kalman filter, the variance in the state estimator is minimized at each time instant given new observations. However, in the case of non-linear dynamical systems, the Extended Kalman Filter (EKF) only minimizes the variance of the linearized state estimate and not the actual state. Particle filters such as condensation ([Isard and Blake, 1998](#)) usually have asymptotic convergence guarantees with respect to the number of particles and the number of time instants. Moreover efficient resampling is needed to deal with cases where all but one particle have non-zero probabilities. Our greedy cost function on the other hand aims to maximize the posterior probability of the state estimate at each time instant by assuming that the previous state is estimated correctly. This might seem like a strong assumption but as our experiments in §5 will show that with the initialization techniques described earlier, we always converge to the correct state.

Parameter Tuning. The variance of the values of individual histogram bins, σ_H^2 , could be empirically computed by using the EM algorithm, given kernel-weighted histograms extracted from training data. However, we fixed the value at $\sigma_H^2 = 0.01$ for all our experiments and this choice consistently gives good tracking performance. The noise parameters, σ_H^2 and Q , can also be analyzed as determining the relative weights of the two terms in the cost function in Eq. (24). The first term in the cost function can be interpreted as a *reconstruction* term that computes the difference between the observed kernel-weighted histogram and the *predicted*

kernel weighted histogram given the state of the system. The second term can similarly be interpreted as a *dynamics* term that computes the difference between the current state and the predicted state given the previous state of the system, regularized by the state-noise covariance. Therefore, the values of σ_H^2 and Q implicitly affect the relative importance of the reconstruction term and the dynamics term in the tracking formulation. As Q is computed during the system identification stage, we do not control the value of this parameter. In fact, if the noise covariance of a particular training system is large, thereby implying less robust dynamic parameters, the tracker will automatically give a low-weight to the dynamics term and a higher one to the reconstruction term.

3.4 Invariance to Scale, Orientation and Direction of Motion

As described at the start of this section, the spatial size, $|\Omega| = r \times c$, of the dynamic template in the training video need not be the same as the size, $|\Omega'| = r' \times c'$, of the template in the test video. Moreover, while tracking, the size of the tracked patch could change from one time instant to the next. For simplicity, we have only considered the case where the size of the patch in the test video stays constant throughout the video. However, to account for a changing patch size, a dynamic model (e.g., a random walk) for $|\Omega'_t| = r'_t \times c'_t$, can easily be incorporated in the derivation of the optimization function in Eq. (24). Furthermore, certain objects such as flags or human actions have a specific direction of motion, and the direction of motion in the training video need not be the same as that in the test video.

To make the tracking procedure of a learnt dynamic object invariant to the size of the selected patch, or the direction of motion, two strategies could be chosen. The first approach is to find a non-linear dynamical systems based representation for dynamic objects that is by design size and pose-invariant, e.g., histograms. This would however pose additional challenges in the gradient descent scheme introduced above and would lead to increased computational complexity. The second approach is to use the proposed LDS-based representation for dynamic objects but transform the system parameters according to the observed size, orientation and direction of motion. We propose to use the second approach and transform the system parameters, μ and C , as required.

Transforming the system parameters of a dynamic texture to model the transformation of the actual dynamic texture was first proposed by [Ravichandran and Vidal \(2011\)](#), where it was noted that two videos of

the same dynamic texture taken from different view points could be registered by computing the *transformation between the system parameters* learnt individually from each video. To remove the basis ambiguity², we first follow Ravichandran and Vidal (2011) and convert all system parameters to the Jordan Canonical Form (JCF). If the system parameters of the training video are $\mathcal{M} = (\mu, A, C, B)$, $\mu \in \mathbb{R}^{rc}$ is in fact the stacked mean template image, $\mu^{\text{im}} \in \mathbb{R}^{r \times c}$. Similarly, we can imagine the matrix $C = [C_1, C_2, \dots, C_n] \in \mathbb{R}^{rc \times n}$ as a composition of basis images $C_i^{\text{im}} \in \mathbb{R}^{r \times c}, i \in \{1, \dots, n\}$.

Given an initialized bounding box around the test patch, we transform the observation parameters μ, C learnt during the training stage to the dimension of the test patch. This is achieved by computing $(\mu')^{\text{im}} = \mu^{\text{im}}(T(x))$ and $(C')^{\text{im}} = C^{\text{im}}(T(x)), i \in 1, \dots, n$, where $T(x)$ is the corresponding transformation on the image domain. For scaling, this transformation is simply an appropriate scaling of the mean image, μ^{im} and the basis images C_i^{im} from $r \times c$ to $r' \times c'$ images using bilinear interpolation. Since the dynamics of the texture of the same type are assumed to be the same, we only need to transform μ and C . The remaining system parameters, A, B , and σ_H^2 , stay the same. For other transformations, such as changes in direction of motion, the corresponding transformation $T(x)$ can be applied to the learnt μ, C , system parameters before tracking. In particular, for human actions, if the change in the direction of motion is simply from left-to-right to right-to-left, μ^{im} , and C^{im} only need to be reflected across the vertical axis to get the transformed system parameters for the test video.

A Note on Discriminative Methods. In the previous development, we have only considered foreground feature statistics. Some state-of-the-art methods also use background feature statistics and adapt the tracking framework according to changes in both foreground and background. For example, Collins et al. (2005a) compute discriminative features such as foreground-to-background feature histogram ratios, variance ratios, and peak difference followed by Meanshift tracking for better performance. Methods based on tracking using classifiers Grabner et al. (2006), Babenko et al. (2009) also build features that best discriminate between foreground and background. Our framework can be easily adapted to such a setting to provide even better performance. We will leave this as future work as our proposed method, based only on foreground statistics,

already provides results similar to or better than the state-of-the-art.

4 Tracking and Recognition of Dynamic Templates

The proposed generative approach presented in §3 has another advantage. As we will describe in detail in this section, we can use the value of the objective function in Eq. (24) to perform simultaneous tracking and recognition of dynamic templates. Moreover, we can learn the LDS parameters of the tracked dynamic template from the corresponding bounding boxes and compute a system-theoretic distance to all the LDS parameters from the training set. This distance can then be used as a discriminative cost to simultaneously provide the best tracks and class label in a test video. In the following we propose three different approaches for tracking and recognition of dynamic templates using the tracking approach presented in the previous section at their core.

Recognition using tracking objective function.

The dynamic template tracker computes the optimal location and state estimate at each time instant by minimizing the objective function in Eq. (24) given the system parameters, $\mathcal{M} = (\mu, A, C, B)$, of the dynamic template, and the kernel histogram variance, σ_H^2 . From here on, we will use the more general expression, $\mathcal{M} = (\mu, A, C, Q, R)$, to describe all the tracker parameters, where $Q = BB^\top$ is the covariance of the state noise process and R is the covariance of the observed image function, e.g., in our proposed kernel-based framework, $R = \sigma_H^2 I$. Given system parameters for multiple dynamic templates, for example, multiple sequences of the same dynamic texture, or sequences belonging to different classes, we can track a dynamic texture in a test video using each of the system parameters. For each tracking experiment, we will obtain location and state estimates for each time instant as well as the value of the objective function at the computed minima. At each time instant, the objective function value computes the value of the negative logarithm of the posterior probability of the location and state given the system parameters. We can therefore compute the average value of the objective function across all frames and use this *dynamic template reconstruction cost* as a measure of how close the observed dynamic template tracks are to the model of the dynamic template used to track it.

More formally, given a set of training system parameters, $\{\mathcal{M}_i\}_{i=1}^N$ corresponding to a set of training videos with dynamic templates with class labels, $\{\mathcal{L}_i\}_{i=1}^N$, and test sequence $j \neq i$, we compute the optimal tracks and states, $\{(\hat{\mathbf{x}}_t^{(j,i)}, \hat{\mathbf{x}}_t^{(j,i)})\}_{t=1}^{T_j}$ for all $i \in 1, \dots, N, j \neq i$ and

² The time series, $\{\mathbf{y}_t\}_{t=1}^T$, can be generated by the system parameters, (μ, A, C, B) , and the corresponding state sequence $\{\mathbf{x}_t\}_{t=1}^T$, or by system parameters $(\mu, PAP^{-1}, CP^{-1}, PB)$, and the state sequence, $\{P\mathbf{x}_t\}_{t=1}^T$. This inherent non-uniqueness of the system parameters given only the observed sequence is referred to as the basis ambiguity.

the corresponding objective function values,

$$\bar{O}_j^i = \frac{1}{T_j} \sum_{t=1}^{T_j} O_j^i(\hat{\mathbf{l}}_t^{(j,i)}, \hat{\mathbf{x}}_t^{(j,i)}), \quad (29)$$

where $O_j^i(\hat{\mathbf{l}}_t^{(j,i)}, \hat{\mathbf{x}}_t^{(j,i)})$ represents the value of the objective function in Eq. (24) computed at the optimal $\hat{\mathbf{l}}_t^{(j,i)}$ and $\hat{\mathbf{x}}_t^{(j,i)}$, when using the system parameters $\mathcal{M}_i = (\mu_i, A_i, C_i, Q_i, R_i)$ corresponding to training sequence i and tracking the template in sequence $j \neq i$. The value of the objective function represents the *dynamic template reconstruction cost* for the test sequence, j , at the computed locations $\{\hat{\mathbf{l}}_t^{(j,i)}\}_{t=1}^{T_j}$ as modeled by the dynamical system \mathcal{M}_i . System parameters that correspond to the dynamics of the same class as the observed template should therefore give the smallest objective function value whereas those that correspond to a different class should give a greater objective function value. Therefore, we can also use the value of the objective function as a classifier to simultaneously determine the class of the dynamic template as well as its tracks as it moves in a scene. The dynamic template class label is hence found as $\mathcal{L}_j = \mathcal{L}_k$, where $k = \operatorname{argmin}_i \bar{O}_j^i$, i.e., the label of the training sequence with the minimum objective function value. The corresponding tracks $\{\hat{\mathbf{l}}_t^{(j,k)}\}_{t=1}^{T_j}$ are used as the final tracking result. Our tracking framework therefore allows us to perform simultaneous tracking and recognition of dynamic objects. We call this method for simultaneously tracking and recognition using the objective function value, Dynamic Kernel SSD - Tracking and Recognition using Reconstruction (**DK-SSD-TR-R**).

Tracking then recognizing. In a more traditional dynamic template recognition framework, it is assumed that the optimal tracks, $\{\hat{\mathbf{l}}_t^{(j,i)}\}_{t=1}^{T_j}$ for the dynamic template have already been extracted from the test video. Corresponding to these tracks, we can extract the sequence of bounding boxes, $Y_j = \{\mathbf{y}_t(\hat{\mathbf{l}}_t^{(j,i)})\}_{t=1}^{T_j}$, and learn the system parameters, \mathcal{M}_j for the tracked dynamic template using the approach described in §3.1. We can then compute a *distance* between the test dynamical system, \mathcal{M}_j , and all the training dynamical systems, $\{\mathcal{M}_i\}, i = 1 \dots N, j \neq i$. A commonly used distance between two linear dynamical systems is the Martin distance, [Cock and Moor \(2002\)](#), that is based on the subspace angles between the observability subspaces of the two systems. The Martin distance has been shown, e.g., in ([Chaudhry and Vidal, 2009](#); [Doretto et al., 2003](#); [Bissacco et al., 2001](#)), to be discriminative between dynamical systems belonging to several different classes. We can therefore use the Martin distance with Nearest Neighbors as a classifier to recognize the test dynamic template by using the optimal tracks, $\{\hat{\mathbf{l}}_t^{(j,k)}\}_{t=1}^{T_j}$, from

the first method, DK-SSD-TR-R. We call this tracking *then* recognizing method Dynamic Kernel SSD - Tracking then Recognizing (**DK-SSD-T+R**).

Recognition using LDS distance classifier. As we will show in the experiments, the *reconstruction cost* based tracking and recognition scheme, DK-SSD-TR-R, works very well when the number of classes is small. However, as the number of classes increases, the classification accuracy decreases. The objective function value itself is in fact not a very good classifier with many classes and high inter class similarity. Moreover, the tracking *then* recognizing scheme, DK-SSD-T+R, disconnects the tracking component from the recognition part. It is possible that tracks that are slightly less optimal according to the objective function criterion may in fact be better for classification. To address this limitation, we propose to add a *classification cost* to our original objective function and use a two-step procedure that computes a distance between the dynamical template as observed through the tracked locations in the test video and the actual training dynamic template. This is motivated by the fact that if the tracked locations in the test video are correct, and a dynamical-systems based distance between the observed template in the test video and the training template is small, then it is highly likely that the tracked dynamic texture in the test video belongs to the same class as the training video. Minimizing a *reconstruction* and *classification* cost will allow us to simultaneously find the best tracks and the corresponding label of the dynamic template.

Assume that with our proposed gradient descent scheme in Eq. (26), we have computed the optimal tracks and state estimates, $\{\hat{\mathbf{l}}_t^{(j,i)}, \hat{\mathbf{x}}_t^{(j,i)}\}_{t=1}^{T_j}$, for all frames in test video j , using the system parameters corresponding to training dynamic template \mathcal{M}_i . As described above, we can then extract the corresponding tracked regions, $Y_j^i = \{\mathbf{y}_t(\hat{\mathbf{l}}_t^{(j,i)})\}_{t=1}^{T_j}$, and learn the system parameters $\mathcal{M}_j^i = (\mu_j^i, A_j^i, C_j^i, Q_j^i)$ using the system identification method in §3.1. If the dynamic template in the observed tracks, \mathcal{M}_j^i , is similar to the training dynamic template, \mathcal{M}_i , then the *distance* between \mathcal{M}_j^i and \mathcal{M}_i should be small. Denoting the Martin distance between two dynamical systems, $\mathcal{M}_1, \mathcal{M}_2$ as $d_M(\mathcal{M}_1, \mathcal{M}_2)$, we propose to use the *classification cost*,

$$C_j^i = d_M(\mathcal{M}_j^i, \mathcal{M}_i). \quad (30)$$

Specifically, we classify the test video as $\mathcal{L}_j = \mathcal{L}_k$, where $k = \operatorname{argmin}_i C_j^i$, and use the extracted tracks, $\{\hat{\mathbf{l}}_t^{(j,k)}\}_{t=1}^{T_j}$ as the final tracks. We call this method for simultaneous tracking and recognition using a classification cost as Dynamic-Kernel SSD - Tracking and Recognition using Classifier (**DK-SSD-TR-C**). As we will show in our ex-

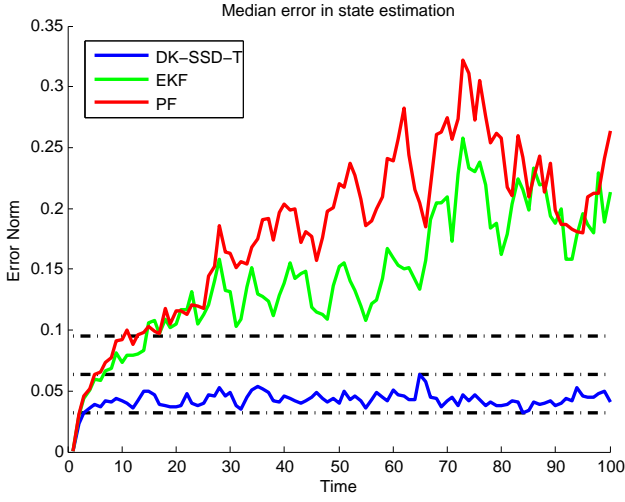


Fig. 5 Median state estimation error across 10 randomly generated dynamic textures with the initial state computed using the pseudo-inverse method in Eq. (28) using our proposed, DK-SSD-T (blue), Extended Kalman Filter (EKF) (green), and Condensation Particle Filter (PF) (red). The horizontal dotted lines represent 1-, 2-, and 3-standard deviations for the norm of the state noise.

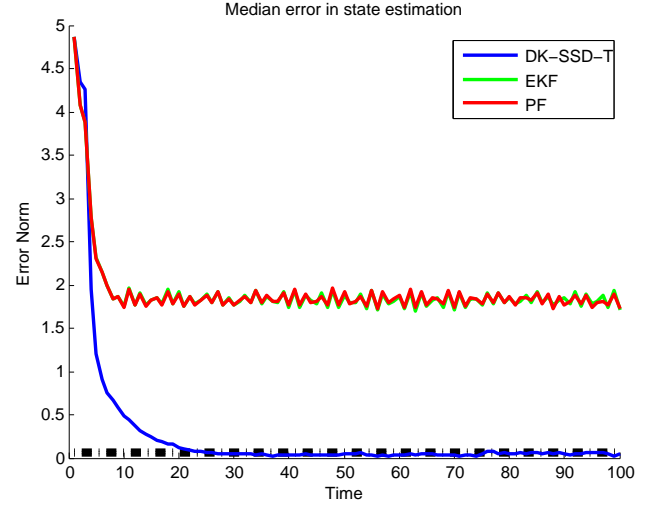
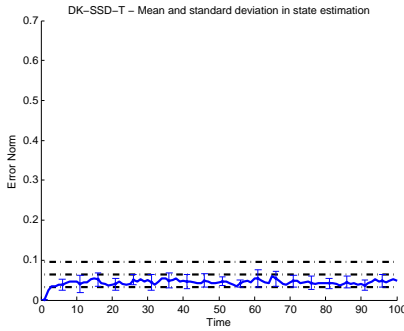
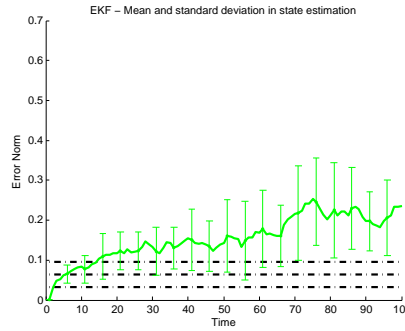


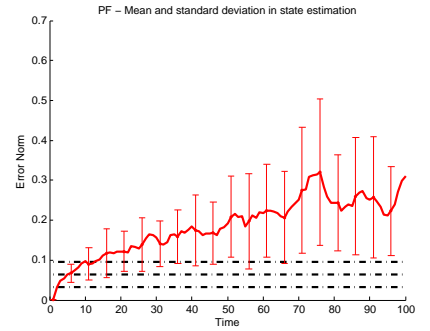
Fig. 6 Median state-error for 10 random initializations for the initial state when estimating the state of the same dynamic texture, using our proposed, DK-SSD-T (blue), Extended Kalman Filter (EKF) (green), and Condensation Particle Filter (PF) (red). The horizontal dotted lines represent 1-, 2-, and 3-standard deviations for the norm of the state noise.



(a) DK-SSD-T

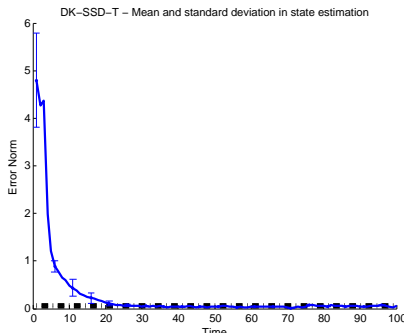


(b) Extended Kalman Filter

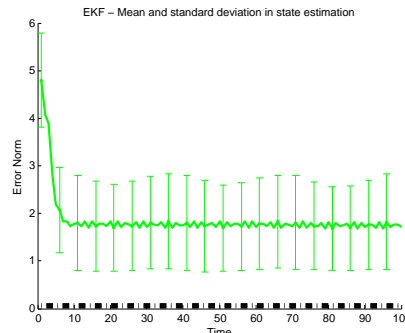


(c) Particle Filter

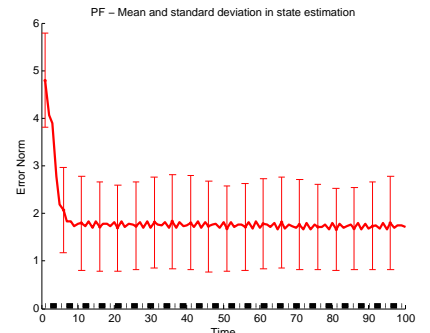
Fig. 7 Mean and 1-standard deviation of state estimation errors for different algorithms across 10 randomly generated dynamic textures with the initial state computed using the pseudo-inverse method in Eq. (28). These figures correspond to the median results shown in Fig. 5.



(a) DK-SSD-T



(b) Extended Kalman Filter



(c) Particle Filter

Fig. 8 Mean and 1-standard deviation of state estimation errors for different algorithms 10 random initializations for the initial state when estimating the state of the same dynamic texture. These figures correspond to the median results shown in Fig. 6.

periments, DK-SSD-TR-C gives state-of-the-art results for tracking and recognition of dynamic templates.

5 Empirical evaluation of state convergence

In §3.3, we discussed the convergence properties of our algorithm. Specifically, we noted that if the template was static or the true output of the dynamic template were known, our proposed algorithm is equivalent to the standard meanshift tracking algorithm. In this section, we will numerically evaluate the convergence of the state estimate of the dynamic template.

We generate a random synthetic dynamic texture with known system parameters and states at each time instant. We then fixed the location of the texture and assumed it was known a-priori thereby reducing the problem to only the estimation of the state given correct measurements. This is also the common scenario for state-estimation in controls theory. Fig. 5 shows the median error in state estimation for 10 randomly generated dynamic textures using the initial state computation method in Eq. (28) in each case. For each of the systems, we estimated the state using our proposed method, Dynamic Kernel SSD Tracking (DK-SSD-T) shown in blue, Extended Kalman Filter (EKF) shown in green, and Condensation Particle Filter (PF), with 100 particles, shown in red, using the same initial state. Since the state, \mathbf{x}_t is driven by stochastic inputs with covariance Q , we also display horizontal bars depicting 1-, 2-, and 3-standard deviations of the norm of the noise process to measure the accuracy of the estimate. As we can see, at all time-instants, the state estimation error using our method remains within 1- and 2-standard deviations of the state noise. The error for both EKF and PF, on the other hand, increases with time and becomes much larger than 3-standard deviations of the noise process.

Figs. 7(a)-7(c) show the mean and standard deviation of the state estimates across all ten runs for DK-SSD-T, EKF and PF respectively. As we can see, our method has a very small standard deviation and thus all runs converge to within 1- and 2-standard deviations of the noise process norm. EKF and PF on the other hand, not only diverge from the true state but the variance in the state estimates also increases with time, thereby making the state estimates very unreliable. This is because our method uses a gradient descent scheme with several iterations to look for the (local) minimizer of the exact objective function, whereas the EKF only uses a linear approximation to the system equations at the current state and does not refine the state estimate any further at each time-step. With a finite number of samples, PF also fails to converge. This leads to a much larger error in the EKF and PF. The trade-off for our

method is its computational complexity. Because of its iterative nature, our algorithm is computationally more expensive as compared to EKF and PF. On average it requires between 25 to 50 iterations for our algorithm to converge to a state estimate.

Similar to the above evaluation, Fig. 6 shows the error in state estimation, for 10 different random initializations of the initial state, $\hat{\mathbf{x}}_0$, for one specific dynamic textures. As we can see, the norm of the state error is very large initially, but for our proposed method, as time proceeds the state error converges to below the state noise error. However the state error for EKF and PF remain very large. Figs. 8(a)-8(c) show the mean and standard deviation bars for the state estimation across all 10 runs. Again, our method converges for all 10 runs, whereas the variance of the state-error is very large for both EKF and PF. These two experiments show that choosing the initial state using the pseudo-inverse method results in very good state estimates. Moreover, our approach is robust to incorrect state initializations and will eventually converge to the correct state with under 2 standard deviations of the state noise.

In summary, the above evaluation shows that even though our method is only guaranteed to converge to a local minimum when estimating the internal state of the system, in practice, it performs very well and always converges to an error within two standard deviations of the state noise. Moreover, our method is robust to incorrect state initializations. Finally, since our method iteratively refines the state estimate at each time instant, it performs much better than traditional state estimation techniques such as the EKF and PF.

6 Experiments on Tracking Dynamic Textures

We will now test our proposed algorithm on several synthetic and real videos with moving dynamic textures and demonstrate the tracking performance of our algorithm against the state-of-the-art. The full videos of the corresponding results can be found at <http://www.cis.jhu.edu/~rizwanch/dynamicTrackingIJCV11> using the figure numbers in this paper.

We will also compare our proposed dynamic template tracking framework against traditional kernel-based tracking methods such as Meanshift (Comaniciu et al., 2003), as well as the improvements suggested in Collins et al. (2005a) that use features such as histogram ratio and variance ratio of the foreground versus the background before applying the standard Meanshift algorithm. We use the publicly available VIVID Tracking Testbed³ (Collins et al., 2005b) for implementations of

³ <http://vision.cse.psu.edu/data/vividEval/>

these algorithms. We also compare our method against the publicly available⁴ Online Boosting algorithm first proposed in Grabner et al. (2006). As mentioned in the introduction, the approach presented in Péteri (2010) also addresses dynamic texture tracking using optical flow methods. Since the authors of Péteri (2010) were not able to provide their code, we implemented their method on our own to perform a comparison. We would like to point out that despite taking a lot of care while implementing, and getting in touch with the authors several times, we were not able to get the same results as those shown in Péteri (2010). However, we are confident that our implementation is correct and besides specific parameter choices, accurately follows the approach proposed in Péteri (2010). Finally, for a fair comparison between several algorithms, we did not use color features and we were able to get very good tracks without using any color information.

For consistency, tracks for Online Boosting (**Boost**) are shown in magenta, Template Matching (**TM**) in yellow, Meanshift (**MS**) in black, Meanshift with Variance Ratio (**MS-VR**) and Histogram Ratio (**MS-HR**) in blue and red respectively. Tracks for Particle Filtering for Dynamic Textures (**DT-PF**) are shown in light brown, and the optimal tracks for our method, Dynamic Kernel SSD Tracking (**DK-SSD-T**), are shown in cyan whereas any ground truth is shown in green. To better illustrate the difference in the tracks, we have zoomed in to the active portion of the video.

6.1 Tracking Synthetic Dynamic Textures

To compare our algorithm against the state-of-the-art on dynamic data with ground-truth, we first create synthetic dynamic texture sequences by manually placing one dynamic texture patch on another dynamic texture. We use sequences from the DynTex database (Péteri et al., 2010) for this purpose. The dynamics of the foreground patch are learnt offline using the method for identifying the parameters, (μ, A, C, B, R) , in Doretto et al. (2003). These are then used in our tracking framework.

In Fig. 9, the dynamic texture is a video of steam rendered over a video of water. We see that Boost, DT-PF, and MS-HR eventually lose track of the dynamic patch. The other methods stay close to the patch however, our proposed method stays closest to the ground truth till the very end. In Fig. 10, the dynamic texture is a sequence of water rendered over a different sequence of water with different dynamics. Here again, Boost and

Algorithm	Fig. 9	Fig. 10
Boost	389 ± 149	82 ± 44
TM	50 ± 13	78 ± 28
MS	12 ± 10	10 ± 8.6
MS-VR	9.2 ± 4.3	3.2 ± 2.0
MS-HR	258 ± 174	4.6 ± 2.3
DT-PF	550 ± 474	635 ± 652
DK-SSD-T	8.6 ± 6.8	6.5 ± 6.6

Table 1 Mean pixel error with standard deviation between tracked location and ground truth.

TM lose tracks. DT-PF stays close to the patch initially but then diverges significantly. The other trackers manage to stay close to the dynamic patch, whereas our proposed tracker (cyan) still performs at par with the best. Fig. 11 also shows the pixel location error at each frame for all the trackers and Table 1 provides the mean error and standard deviation for the whole video. Overall, MS-VR and our method seem to be the best, although MS-VR has a lower standard deviation in both cases. However note that our method gets similar performance without the use of background information, whereas MS-VR and MS-HR use background information to build more discriminative features. Due to the dynamic changes in background appearance, Boost fails to track in both cases. Even though DT-PF is designed for dynamic textures, upon inspection, all the particles generated turn out to have the same (low) probability. Therefore, the tracker diverges. Given the fact that our method is only based on the appearance statistics of the foreground patch, as opposed to the adaptively changing foreground/background model of MS-VR, we attribute the comparable performance (especially against other foreground-only methods) to the explicit inclusion of foreground dynamics.

6.2 Tracking Real Dynamic Textures

To test our algorithm on real videos with dynamic textures, we provide results on three different scenarios in real videos. We learn the system parameters of the dynamic texture by marking a bounding box around the texture in a video where the texture is stationary. We then use these parameters to track the dynamic texture in a different video with camera motion causing the texture to move around in the scene. All the trackers are initialized at the same location in the test video. Fig. 12 provides the results for the three examples by alternatively showing the training video followed by the tracker results in the test video. We will use (row, column) notation to refer to specific images in Fig. 12.

Candle Flame. This is a particularly challenging scene as there are multiple candles with similar dynamics and

⁴ <http://www.vision.ee.ethz.ch/boostingTrackers/index.htm>

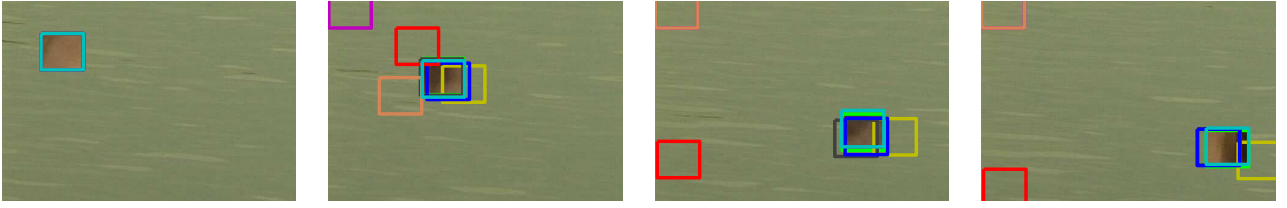


Fig. 9 Tracking steam on water waves. [Boost (magenta), TM (yellow), MS (black), MS-VR (blue) and MS-HR (red), DT-PF (light brown), DK-SSD-T (cyan). Ground-truth (green)]



Fig. 10 Tracking water with different appearance dynamics on water waves. [Boost (magenta), TM (yellow), MS (black), MS-VR (blue) and MS-HR (red), DT-PF (light brown), DK-SSD-T (cyan). Ground-truth (green)]

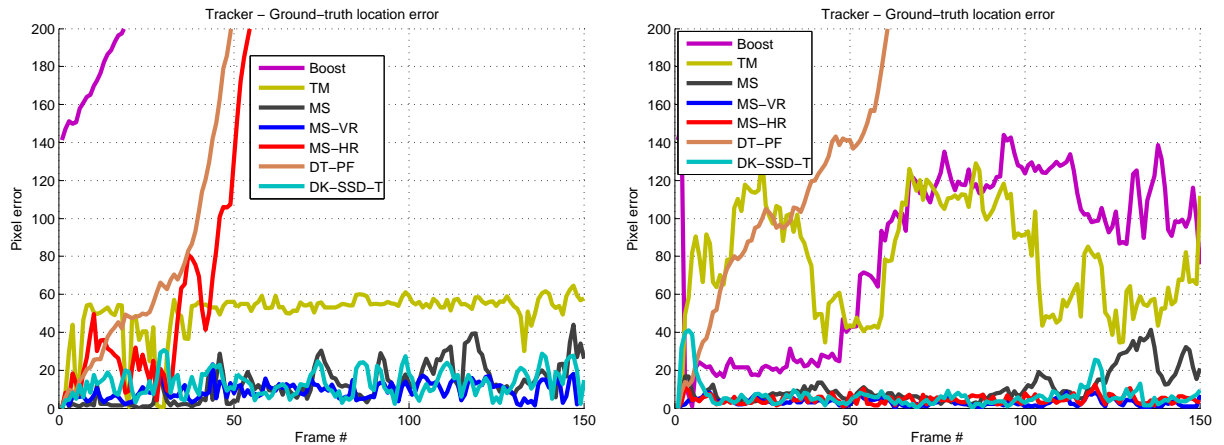


Fig. 11 Pixel location error between tracker and ground truth location for videos in Fig. 9 (left) and Fig. 10 (right).

appearance and it is easy for a tracker to switch between candles. As we can see from (2,2), MS-HR and MS-VR seem to jump around between candles. Our method also jumps to another candle in (2,3) but recovers in (2,4), whereas MS is unable to recover. DT-PF quickly loses track and diverges. Overall, all trackers, except DT-PF seem to perform equally well.

Flags. Even though the flag has a distinct appearance compared to the background, the movement of the flag fluttering in the air changes the appearance in a dynamic fashion. Since our tracker has learnt an explicit model of the dynamics of these appearance changes, it stays closest to the correct location while testing. Boost, DT-PF, and the other trackers deviate from the true location as can be seen in (4,3) and (4,4).

Fire. Here we show another practical application of our proposed method: tracking fire in videos. We learn a dynamical model for fire from a training video which is taken in a completely different domain, e.g., a campfire. We then use these learnt parameters to track fire in a NASCAR video as shown in the last row of Fig. 12. Our foreground-only dynamic tracker performs better than MS and TM, the other foreground-only methods in comparison. Boost, MS-VR and MS-HR use background information, which is discriminative enough in this particular video and achieve similar results. DT-PF diverges from the true location and performs the worst.



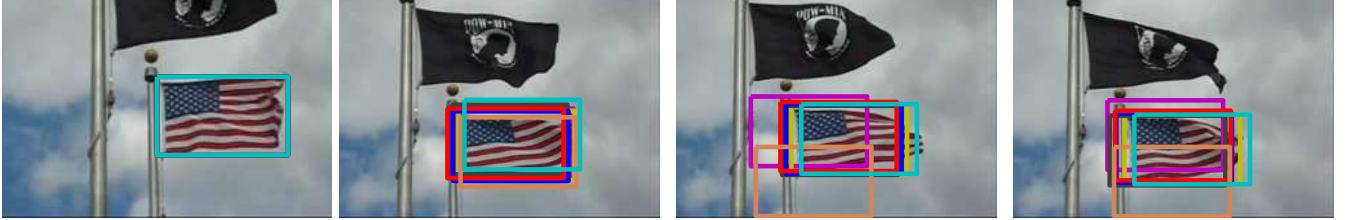
(a) Training video with labeled stationary patch for learning *candle flame* dynamic texture system parameters.



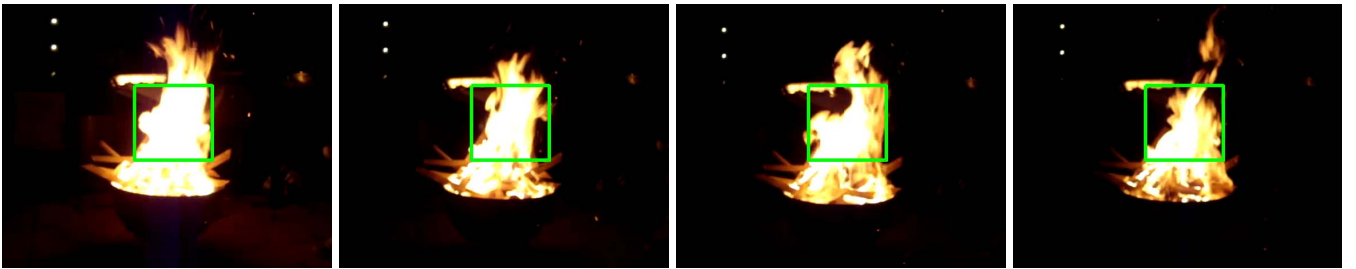
(b) Test video with tracked locations of *candle flame*.



(c) Training video with labeled stationary patch for learning *flag* dynamic texture system parameters.



(d) Test video with tracked locations of *flag*.



(e) Training video with labeled stationary patch for learning *fire* dynamic texture system parameters.



(f) Test video with tracked locations of *fire*.

Fig. 12 Training and Testing results for dynamic texture tracking. Boost (magenta), TM (yellow), MS (black), MS-VR (blue) and MS-HR (red), DT-PF (light brown), DK-SSD-T (cyan). Training (green).

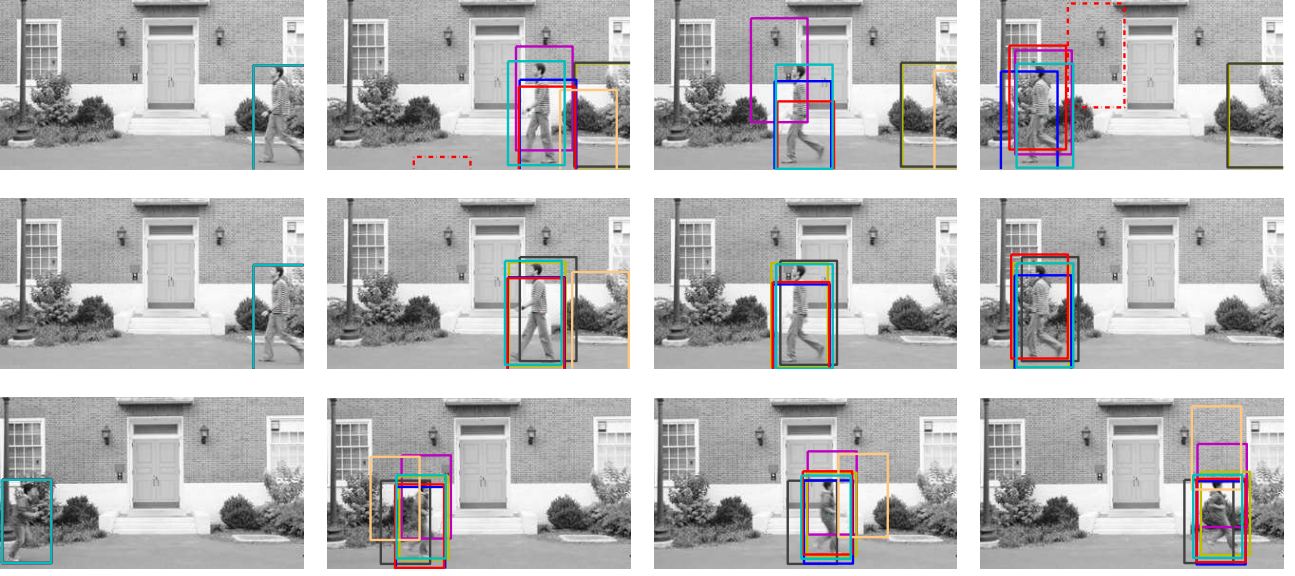


Fig. 13 Using the dynamics of optical flow to track a human action. Parts-based human detector (red broken), Boost (magenta), TM (yellow), MS (black), MS-VR (blue) and MS-HR (red), DK-SSD-T (cyan). First row: Tracking a walking person, without any pre-processing for state-of-the-art methods. Second row: Tracking a walking person, with pre-processing for state-of-the-art methods. Third row: Tracking a running person, with pre-processing for state-of-the-art methods.

7 Experiments on Tracking Human Actions

To demonstrate that our framework is general and can be applied to track dynamic visual phenomenon in any domain, we consider the problem of tracking humans while performing specific actions. This is different from the general problem of tracking humans, as we want to track humans performing specific actions such as walking or running. It has been shown, e.g., in [Efros et al. \(2003\)](#); [Chaudhry et al. \(2009\)](#); [Lin et al. \(2009\)](#) that the optical flow generated by the motion of a person in the scene is characteristic of the action being performed by the human. In general global features extracted from optical flow perform better than intensity-based global features for action recognition tasks. The variation in the optical flow signature as a person performs an action displays very characteristic dynamics. We therefore model the variation in optical flow generated by the motion of a person in a scene as the output of a linear dynamical system and pose the action tracking problem in terms of matching the observed optical flow with a dynamic template of flow fields for that particular action.

We collect a dataset of 55 videos, each containing a single human performing either of two actions (walking and running). The videos are taken with a stationary camera, however the background has a small amount of dynamic content in the form of waving bushes. Simple background subtraction would therefore lead to erroneous bounding boxes. We manually extract bounding boxes and the corresponding centroids to mark the loca-

tion of the person in each frame of the video. These are then used as ground-truth for later comparisons as well as to learn the dynamics of the optical flow generated by the person as they move in the scene.

For each bounding box centered at \mathbf{l}_t , we extract the corresponding optical flow $\mathcal{F}(\mathbf{l}_t) = [F_x(\mathbf{l}_t), F_y(\mathbf{l}_t)]$, and model the optical flow time-series, $\{\mathcal{F}(\mathbf{l}_t)\}_{t=1}^T$ as a Linear Dynamical System. We extract the system parameters, (μ, A, C, Q, R) for each optical flow time-series using the system identification method in §3.1. This gives us the system parameters and ground-truth tracks for each of the 55 human action samples.

Given a test video, computing the tracks and internal state of the optical-flow dynamical system at each time instant amounts to minimizing the function,

$$O(\mathbf{l}_t, \mathbf{x}_t) = \frac{1}{2R} \|\mathcal{F}(\mathbf{l}_t) - (\mu + C\mathbf{x}_t)\|^2 + \frac{1}{2}(\mathbf{x}_t - A\mathbf{x}_{t-1})^\top Q^{-1}(\mathbf{x}_t - A\mathbf{x}_{t-1}). \quad (31)$$

To find the optimal \mathbf{l}_t , and \mathbf{x}_t , Eq. (31) is optimized in the same gradient descent fashion as Eq. (24).

We use the learnt system parameters in a leave-one-out fashion to track the activity in each test video sequence. Taking one sequence as a test sequence, we use all the remaining sequences as training sequences. The flow-dynamics system parameters extracted from each training sequence are used to track the action in the test sequence. Therefore, for each test sequence $j \in \{1, \dots, N\}$, we get $N - 1$ tracked locations and

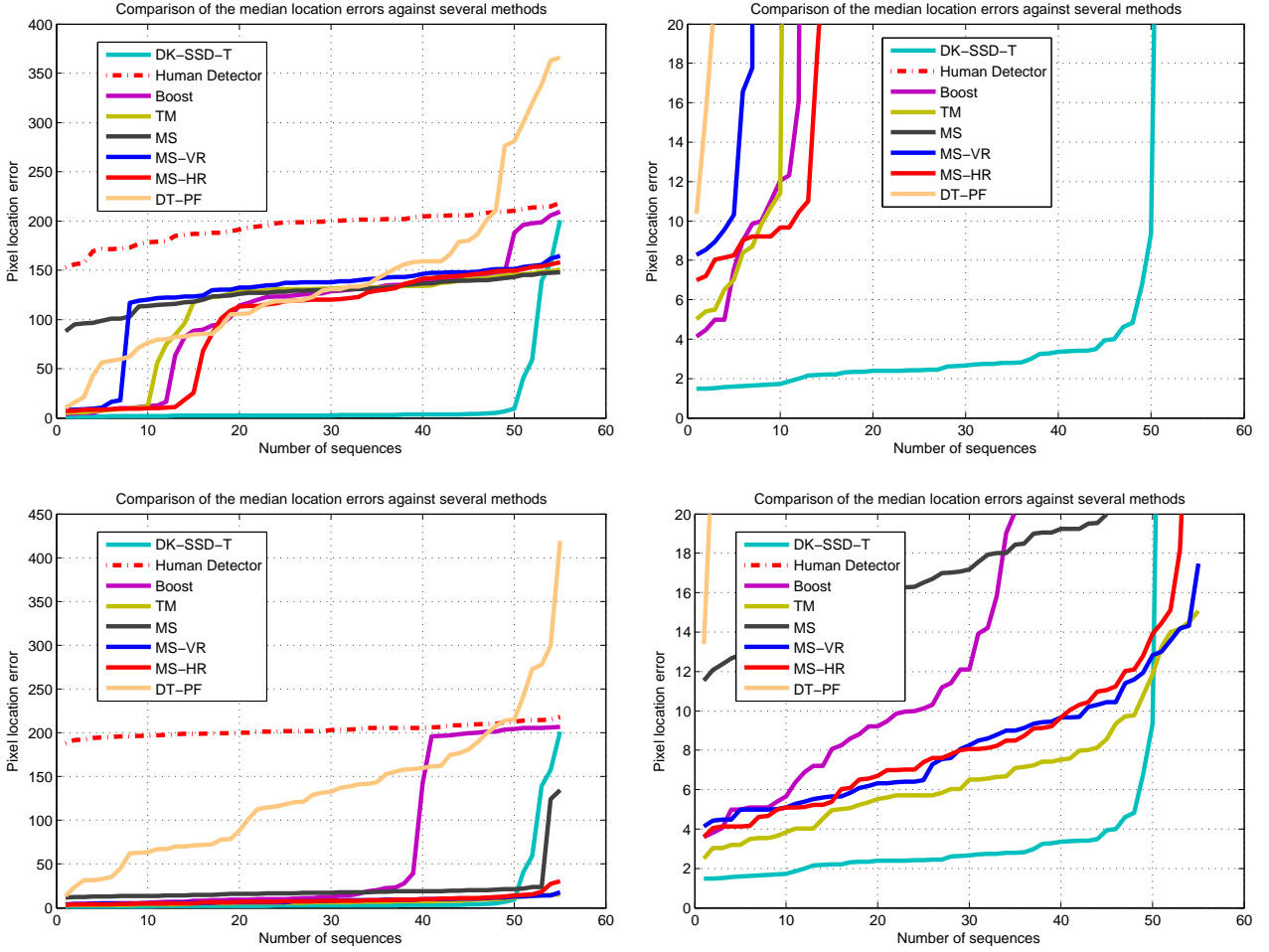


Fig. 14 Comparison of various tracking methods without (top row) and with (bottom row) background subtraction. Figures on right are zoomed-in versions of figures on left.

state estimate time-series by using all the remaining $N - 1$ extracted system parameters. As described in §4, we choose the tracks that give the minimum objective function value in Eq. (29).

Fig. 13 shows the tracking results against the state-of-the-art algorithms. Since this is a human activity tracking problem, we also compare our method against the parts-based human detector of Felzenszwalb et al. (2010) when trained on the PASCAL human dataset. We used the publicly available⁵ code for this comparison with default parameters and thresholds⁶. The detection with the highest probability is used as the location of the human in each frame.

The first row in Fig. 13 shows the results for all the trackers when applied to the test video. The state-of-the-art trackers do not perform very well. In fact

⁵ <http://people.cs.uchicago.edu/~pff/latent/>

⁶ It might be possible to achieve better detections on this dataset by tweaking parameters/thresholds. However we did not attempt this as it is not the focus of our paper.

foreground-only trackers, MS, TM and DT-PF, lose tracks altogether. Our proposed method (cyan) gives the best tracking results and the best bounding box covering the person across all frames. The parts-based detector at times does not give any responses or spurious detections altogether, whereas Boost, MS-VR and MS-HR do not properly align with the human. Since we use optical flow as a feature, it might seem that there is an implicit form of background subtraction in our method. As a more fair comparison, we performed background subtraction as a pre-processing step on all the test videos before using the state-of-the-art trackers. The second row in Fig. 13 shows the tracking results for the same walking video as in row 1. We see that the tracking has improved but our tracker still gives the best results. The third row in Fig. 13 shows tracking results for a running person. Here again, our tracker performs the best whereas Boost and DT-PF perform the worst.

To derive quantitative conclusions about tracker performance, Fig. 14 shows the median tracker location

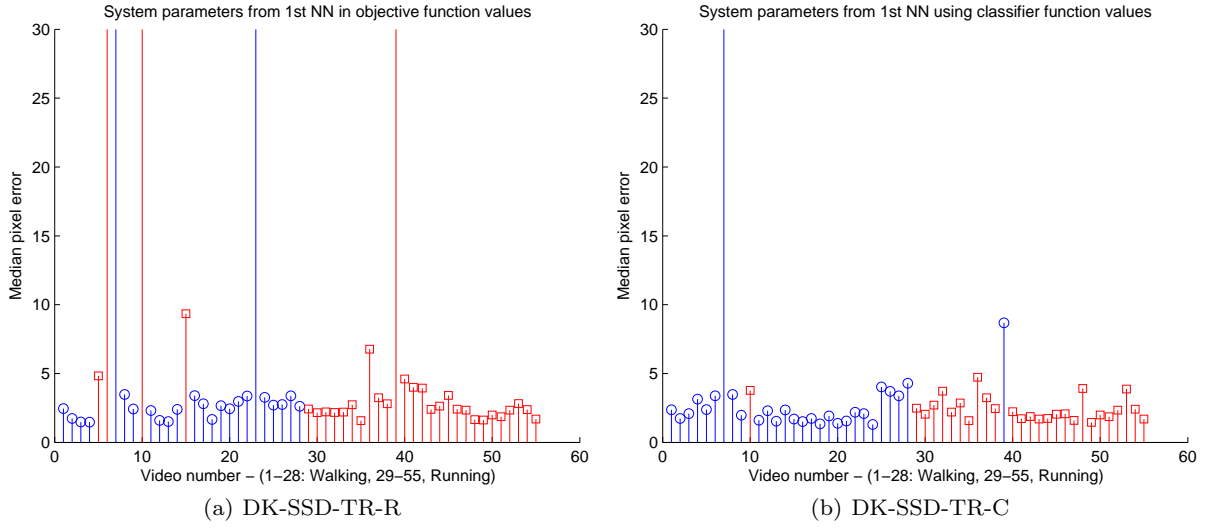


Fig. 15 Simultaneous Tracking and Recognition results showing median tracking error and classification results, when using (a) DK-SSD-TR-R: the objective function in Eq. (29), and (b) DK-SSD-TR-C: the Martin distance between dynamical systems in Eq. (30) with a 1-NN classifier. (walking (blue), running (red)).

error for each video sorted in ascending order. The best method should have the smallest error for most of the videos. As we can see, both without (1st row) and with background subtraction (2nd row), our method provides the smallest median location error against all state of the art methods for all except 5 sequences. Moreover, as a black box and without background subtraction as a pre-processing step, all state-of-the-art methods perform extremely poorly.

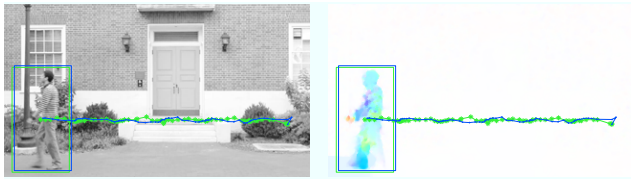
8 Experiments on Simultaneous Action Tracking and Recognition

In the previous section, we have shown that given training examples for actions, our tracking framework, DK-SSD-T, can be used to perform human action tracking using system parameters learnt from training data with correct tracks. We used the value of the objective function in Eq. (31) to select the tracking result. In this section, we will extensively test our simultaneous tracking and classification approach presented in §4 on the two-action database introduced in the previous section as well as the commonly used Weizmann human action dataset (Gorelick et al., 2007). We will also show that we can learn the system parameters for a class of dynamic templates from one database and use it to simultaneously track and recognize the template in novel videos from other databases.

8.1 Walking/Running Database

Fig. 15 shows the median pixel tracking error of each test sequence using the leave-one-out validation described in the previous section for selecting the tracks, sorted by true action class. The first 28 sequences belong to the class *walking*, while the remaining 27 sequences belong to the class *running*. Sequences identified by our proposed approach as walking are colored blue, whereas sequences identified as running are colored red. Fig. 15(a) shows the tracking error and classification result when using DK-SSD-TR-R, i.e., the objective function value in Eq. (29) with a 1-NN classifier to simultaneously track and recognize the action. The tracking results shown also correspond to the ones shown in the previous section. As we can see, for almost all sequences, the tracking error is within 5 pixels from the ground truth tracks. Moreover, for all but 4 sequences, the action is classified correctly, leading to an overall action classification rate of 93%. Fig. 15(b) shows the tracking error and class labels when we use DK-SSD-TR-C, i.e., the Martin distance based classifier term proposed in Eq. (30) to simultaneously track and recognize the action. As we can see, DK-SSD-TR-C results in even better tracking and classification results. Only two sequences are mis-classified for an overall recognition rate of 96%.

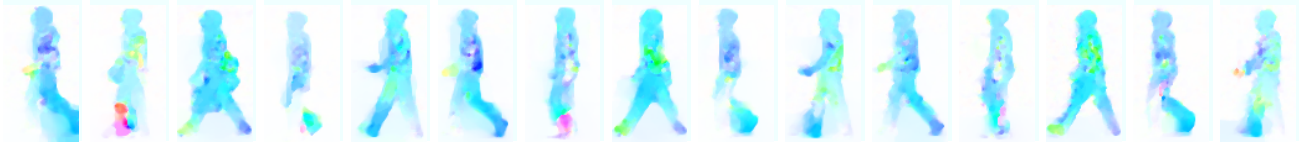
To show that our simultaneous tracking and classification framework computes the correct state of the dynamical system for the test video given that the class of the training system is the same, we illustrate several components of the action tracking framework for two cases: a right to left walking person tracked using dy-



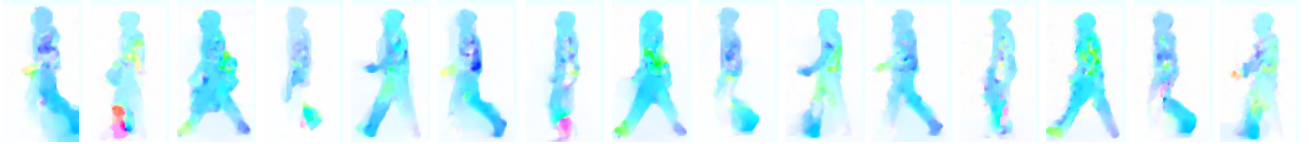
(a) Intensity and optical flow frame with ground-truth tracks (green) and tracked outputs of our algorithm (blue) on test video with walking person tracked using system parameters learnt from the walking person video in (b).



(b) Intensity and optical flow frame with ground-truth tracks (green) used to train system parameters for a walking model.



(c) Optical flow bounding boxes at ground truth locations.



(d) Optical flow bounding boxes at tracked locations.



(e) Optical flow as generated by the computed states at corresponding time instants at the tracked locations.



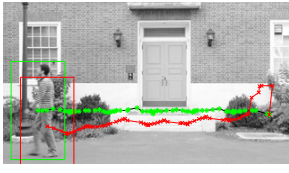
(f) Optical flow bounding boxes from the training video used to learn system parameters.

Fig. 16 Tracking a walking person using dynamical system parameters learnt from another walking person with opposite walking direction. The color of the optical flow diagrams represents the direction (e.g., right to left is cyan, right to left is red) and the intensity represents the magnitude of the optical flow vector.

namical system parameters learnt from another walking person moving left to right, in Fig. 16, and the same person tracked using dynamical system parameters learnt from a running person in Fig. 17.

Fig. 16(a) shows a frame with its corresponding optical flow, ground truth tracks (green) as well as the tracks computed using our algorithm (blue). As we can see the computed tracks accurately line-up with the ground-truth tracks. Fig. 16(b) shows a frame and corresponding optical flow along with the ground-truth tracks used to extract optical flow bounding boxes to learn the dynamical system parameters. Fig. 16(c) shows the optical flow extracted from the bounding boxes at the ground-truth locations in the test video at intervals of 5 frames and Fig. 16(d) shows the optical flow extracted

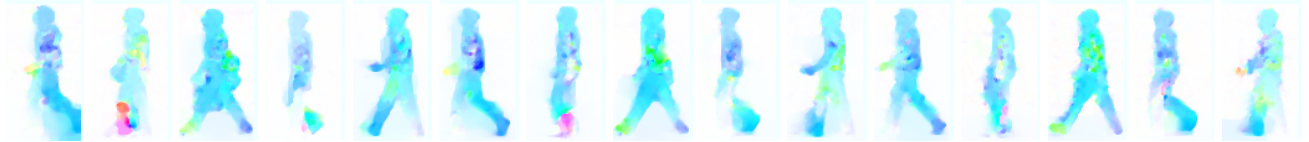
from the bounding boxes at the tracked locations at the same frame numbers. As the extracted tracks are very accurate, the flow-bounding boxes line up very accurately. Since our dynamical system model is generative, at each time-instant, we can use the computed state, $\hat{\mathbf{x}}_t$, to generate the corresponding output $\hat{\mathbf{y}}_t = \mu + C\hat{\mathbf{x}}_t$. Fig. 16(e) displays the optical flow computed in this manner at the corresponding frames in Fig. 16(d). We can see that the generated flow appears like a smoothed version of the observed flow at the correct location. This shows that the internal state of the system was correctly computed according to the training system parameters, which leads to accurate dynamic template generation and tracking. Fig. 16(f) shows the optical flow at the bounding boxes extracted from the ground-truth loca-



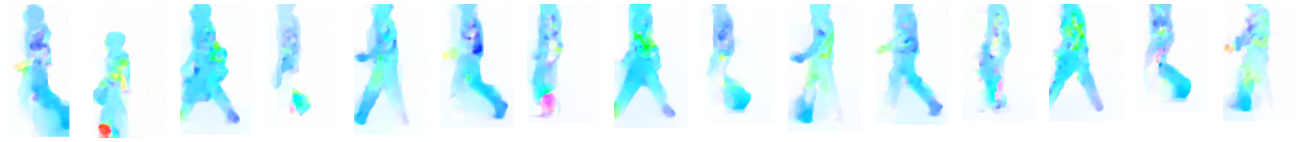
(a) Intensity and optical flow frame with ground-truth tracks (green) and tracked outputs of our algorithm (red) on test video with walking person tracked using system parameters learnt from the running person video in (b).



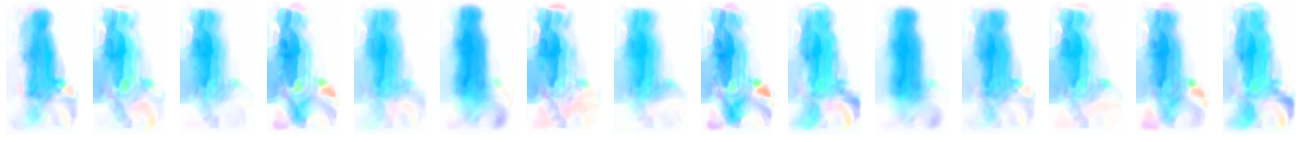
(b) Intensity and optical flow frame with ground-truth tracks (green) used to train system parameters for a running model.



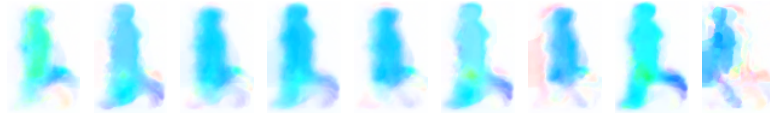
(c) Optical flow bounding boxes at ground truth locations.



(d) Optical flow bounding boxes at tracked locations.



(e) Optical flow as generated by the computed states at corresponding time instants at the tracked locations.



(f) Optical flow bounding boxes from the training video used to learn system parameters.

Fig. 17 Tracking a walking person using dynamical system parameters learnt from a running person. The color of the optical flow diagrams represents the direction (e.g., right to left is cyan, right to left is red) and the intensity represents the magnitude of the optical flow vector.

tions in the training video. The direction of motion is the opposite as in the test video, however using the mean optical flow direction, the system parameters can be appropriately transformed at test time to account for this change in direction as discussed in §3.4.

Fig. 17(a) repeats the above experiment when tracking the same walking person with a running model. As we can see in Fig. 17(a), the computed tracks are not very accurate when using the wrong action class for tracking. This is also evident in the extracted flow bounding boxes at the tracked locations as the head of the person is missing from almost all of the boxes. Fig. 17(f) shows several bounding box optical flow from the training video of the running person. The states computed using the learnt dynamical system parameters from these bounding boxes leads to the generated

flows in Fig. 17(e) at the frames corresponding to those in Fig. 17(d). The generated flow does not match the ground-truth flow and leads to a high objective function value.

8.2 Comparison to Tracking then Recognizing

We will now compare our simultaneous tracking and recognition approaches, DK-SSD-TR-R and DK-SSD-TR-C to the tracking *then* recognizing approach where we first track the action using standard tracking algorithms described in §6 as well as our proposed dynamic tracker DK-SSD-T, and then use the Martin distance for dynamical systems with a 1-NN classifier to classify the tracked action.

Given a test video, we compute the location of the person using all the trackers described in §6 and then extract the bounding box around the tracked locations. For bounding boxes that do not cover the image area, we zero-pad the optical flow. This gives us an optical-flow time-series corresponding to the extracted tracks. We then use the approach described in §3.1 to learn the dynamical system parameters of this optical flow time-series. To classify the tracked action, we compute the Martin distance of the tracked system to all the training systems in the database and use 1-NN to classify the action. We then average the results over all sequences in a leave-one-out fashion.

Table 2 shows the recognition results for the 2-class Walking/Running database by using this classification scheme after performing tracking as well as our proposed simultaneous tracking and recognition algorithms. We have provided results for both the original sequences as well as when background subtraction was performed prior to tracking. We showed in Fig. 14(a) and Fig. 14(c) that all the standard trackers performed poorly without using background subtraction. Our tracking *then* recognizing method, DK-SSD-T+R, provides accurate tracks and therefore gives a recognition rate of 96.36%. The parts-based human detector of Felzenszwalb et al. (2010) tracker fails to detect the person in some frames and therefore the recognition rate is the worst. When using background subtraction to pre-process the videos, the best recognition rate is provided by TM at 98.18% whereas MS-HR performs at the same rate as our proposed method. The recognition rate of the other methods, except the human detector, also increase due to this pre-processing step. For comparison, if we use the ground-truth tracks to learn the dynamical system parameters and classify the action using the Martin distance and 1-NN classification, we get 100% recognition.

Our joint-optimization scheme using the objective function value as the classifier, DK-SSD-TR-R, performs slightly worse at 92.73%, than the best tracking *then* recognizing approach. However, when we use the Martin-distance based classification cost in DK-SSD-TR-C, we get the best action classification performance, 96.36%. Needless to say the tracking *then* recognizing scheme is more computationally intensive and is a 2-step procedure. Moreover, our joint optimization scheme based only on foreground dynamics performs better than tracking *then* recognizing using all other trackers without pre-processing. At this point, we would also like to note that even though the best recognition percentage achieved by the tracking-then-recognize method was 98.18% using TM, it performed worse in tracking as shown in Fig.

Method	Recognition % without background subtraction	Recognition % with background subtraction
Track <i>then</i> recognize		
Boost	81.82	81.82
TM	78.18	98.18
MS	67.27	89.09
MS-VR	74.55	94.55
MS-HR	78.18	96.36
Human-detector	45.45	50.91
DT-PF	69.09	72.73
DK-SSD-T+R	96.36	-
Ground-truth tracks	100	-
Joint-optimization		
DK-SSD-TR-R	92.73	-
DK-SSD-TR-C	96.36	-

Table 2 Recognition rates for the 2-class Walking/Running database using 1-NN with Martin distance for dynamical systems after computing tracks from different algorithms

Action	Median \pm RSE	Action	Median \pm RSE
Bend	5.7 ± 1.9	Side	2.1 ± 0.9
Jack	4.2 ± 0.6	Skip	2.8 ± 1.1
Jump	1.5 ± 0.2	Walk	3.3 ± 2.4
PJump	3.7 ± 1.3	Wave1	14.8 ± 7.4
Run	4.2 ± 2.9	Wave2	2.3 ± 0.7

Table 3 Median and robust standard error (RSE) (see text) of the tracking error for the Weizmann Action database, using the proposed simultaneous tracking and recognition approach with the classification cost. We get an overall action recognition rate of 92.47%.

14(d). Overall, our method simultaneously gives the best tracking and recognition performance.

8.3 Weizmann Action Database

We also tested our simultaneous tracking and testing framework on the Weizmann Action database (Gorelick et al., 2007). This database consists of 10 actions with a total of 93 sequences and contains both stationary actions such as jumping in place, bending etc., and non-stationary actions such as running, walking, etc. We used the provided backgrounds to extract bounding boxes and ground-truth tracks from all the sequences and learnt the parameters of the optical-flow dynamical systems using the same approach as outlined earlier. A commonly used evaluation scheme for the Weizmann database is leave-one-out classification. Therefore, we also used our proposed framework to track the action in a test video given the system parameters of all the remaining actions.

Table 3 shows the median and robust standard error (RSE), i.e., the square-root of the median of $\{(x - \text{median}(x))^2\}$, of the tracking error for each class. Other

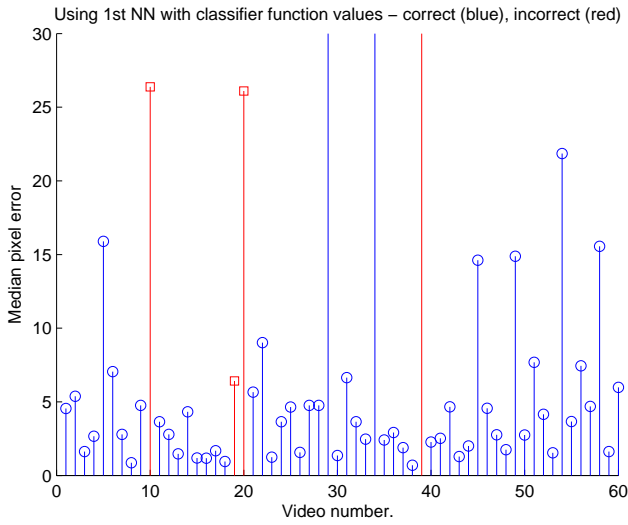


Fig. 18 Simultaneous Tracking and Recognition results showing median tracking error and classification results, when using the objective function, and when using the Martin distance between dynamical systems with a 1-NN classifier. (walking (blue), running (red)).

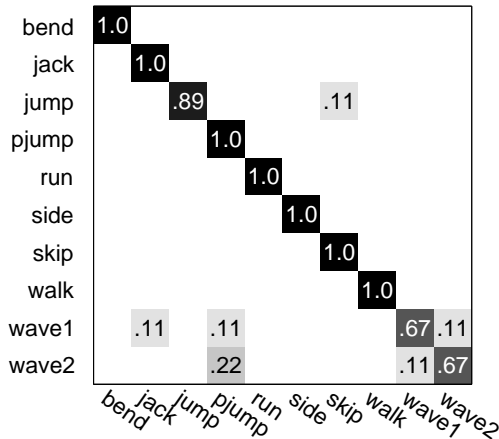


Fig. 19 Confusion matrix for leave-one-out classification on the Weizmann database using our simultaneous tracking and recognition approach. Overall recognition rate of 92.47%.

than Wave1, all classes have a median tracking error under 6 pixels as well as very small deviations from the median error. Furthermore, we get a simultaneous recognition rate of 92.47% which corresponds to only 7 mis-classified sequences. Fig. 19 shows the corresponding confusion matrix. Fig. 18 shows the median tracking error per frame for each of the 93 sequences in the Weizmann database. The color of the stem-plot indicates whether the sequence was classified correctly (blue) or incorrectly (red). Table 4 shows the recognition rate of some state-of-the-art methods on the Weizmann database. However notice that, all these methods are geared towards recognition and either assume that tracking has been accurately done before the recogni-

Method	Recognition (%)
Xie et al. (2011)	95.60
Thurau and Hlavac (2008)	94.40
Ikizler and Duygulu (2009)	100.00
Gorelick et al. (2007)	99.60
Niebles et al. (2008)	90.00
Ali and Shah (2010)	95.75
Ground-truth tracks (1-NN Martin)	96.77
Our method, DK-SSD-TR-C	92.47

Table 4 Comparison of different approaches for action recognition on the Weizmann database against our simultaneous tracking and recognition approach.

tion is performed, or use spatio-temporal features for recognition that can not be used for accurate tracking. Furthermore, if the ground-truth tracks were provided, using the Martin distance between dynamical systems with a 1-NN classifier gives a recognition rate of 96.77%. Our simultaneous tracking and recognition approach is very close to this performance. The method by Xie et al. (2011) seems to be the only attempt to simultaneously locate and recognize human actions in videos. However their method does not perform tracking, instead it extends the parts-based detector by Felzenszwalb et al. (2010) to explicitly consider temporal variations caused by various actions. Moreover, they do not have any tracking results in their paper other than a few qualitative detection results. Our approach is the first to explicitly enable simultaneous tracking and recognition of dynamic templates that is generalizable to any dynamic visual phenomenon and not just human actions.

We will now demonstrate that our simultaneous tracking and recognition framework is fairly general and we can train for a dynamic template on one database and use the models to test on a totally different database. We used our trained walking and running action models (i.e., the corresponding system parameters) from the 2-class walking/running database in §8.1 and applied our proposed algorithm for joint tracking and recognizing the running and walking videos in the Weizmann human action database (Gorelick et al., 2007), without any a-priori training or adapting to the Weizmann database. Of the 93 videos in the Weizmann database, there are a total of 10 walking and 10 running sequences. Fig. 20 shows the median tracking error for each sequence and the color codes its action, running (red) and walking (blue). As we can see, all the sequences have under 5 pixel median pixel error and all the 20 sequences are classified correctly. This demonstrates the fact that our scheme is general and that the requirement of learning the system parameters of the dynamic template is not necessarily a bottleneck as the parameters need not be learnt again for every scenario. We can use the learnt

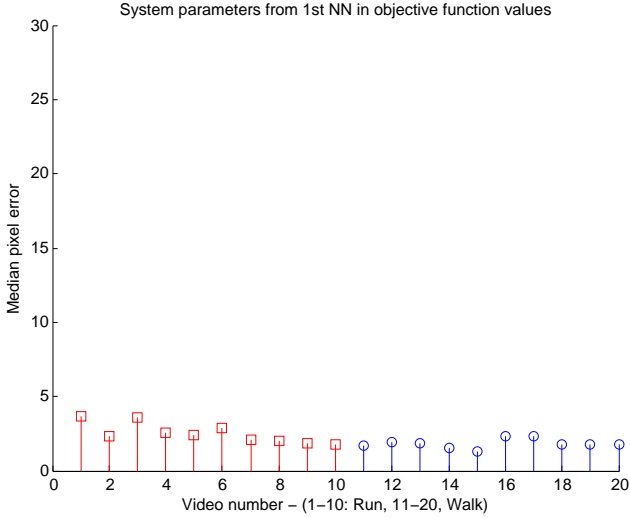


Fig. 20 Tracking walking and running sequences in the Weizmann database using trained system parameters from the database introduced in §8.1. The ground-truth label of the first 10 sequences is running, while the rest are walking. The result of the joint tracking and recognition scheme are labeled as running (red) and walking (blue). We get 100% recognition and under 5 pixel median location error.

system parameters from one dataset to perform tracking and recognition in a different dataset.

9 Conclusions, Limitations and Future Work

In this paper, we have proposed a novel framework for tracking dynamic templates such as dynamic textures and human actions that are modeled by Linear Dynamical Systems. We posed the tracking problem as a maximum a-posteriori estimation problem for the current location and the LDS state, given the current image features and the previous state. By explicitly considering the dynamics of only the foreground, we are able to get state-of-the-art tracking results on both synthetic and real video sequences against methods that use additional information about the background. Moreover we have shown that our approach is general and can be applied to any dynamic feature such as optical flow. Our method performs at par with state-of-the-art methods when tracking human actions. We have shown excellent results for simultaneous tracking and recognition of human actions and demonstrated that our method performs better than simply tracking then recognizing human actions when no pre-processing is performed on the test sequences. However our approach is computationally more efficient as it provides both tracks and template recognition at the same cost. Finally, we showed that the requirement of having a training set of system parameters for the dynamic templates is not re-

strictive as we can train on one dataset and then use the learnt parameters at test time on any sequence where the desired action needs to be found.

Although our simultaneous tracking and recognition approach has shown promising results, there are certain limitations. Firstly, as mentioned earlier, since our method uses gradient descent, it is amenable to converge to non-optimal local minima. Having a highly non-linear feature function or non-linear dynamics could potentially result in sub-optimal state estimation which could lead to high objective function values even when the correct class is chosen for tracking. Therefore, if possible, it is better to choose linear dynamics and model system parameter changes under different transformations instead of modeling dynamics of transformation-invariant but highly non-linear features. However it must be noted that for robustness, some non-linearities in features such as kernel-weighted histograms are necessary and need to be modeled. Secondly since our approach requires performing tracking and recognition using all the training data, it could be computationally expensive as the number of training sequences and the number of classes increases. This can be alleviated by using a smart classifier or more generic one-model-per-class based methods. We leave this as future work.

In other future work, we are looking at online learning of the system parameters of the dynamic template, so that for any new dynamic texture in the scene, the system parameters are also learnt simultaneously as the tracking proceeds. This way, our approach will be applicable to dynamic templates for which system parameters are not readily available.

Acknowledgements The authors would like to thank Diego Rother for useful discussions. This work was supported by the grants, NSF 0941463, NSF 0941362 and ONR N00014-09-10084.

A Derivation of the Gradient Descent Scheme

In this appendix, we will show the detailed derivation of the iterations Eq. (26) to minimize the objective function in Eq. (24), with ρ , the non-differentiable kernel weighted histogram replaced by ζ , our proposed differentiable kernel weighted histogram:

$$O(\mathbf{l}_t, \mathbf{x}_t) = \frac{1}{2\sigma_H^2} \|\sqrt{\zeta(\mathbf{y}_t(\mathbf{l}_t))} - \sqrt{\zeta(\mu + C\mathbf{x}_t)}\|^2 + \frac{1}{2} (\mathbf{x}_t - A\hat{\mathbf{x}}_{t-1})^\top Q^{-1} (\mathbf{x}_t - A\hat{\mathbf{x}}_{t-1}). \quad (32)$$

Using the change in variable, $\mathbf{z}' = \mathbf{z} + \mathbf{l}_t$, the proposed kernel weighted histogram for bin u , Eq. (25),

$$\zeta_u(\mathbf{y}_t(\mathbf{l}_t)) = \frac{1}{\kappa} \sum_{\mathbf{z} \in \Omega} K(\mathbf{z}).$$

$$(\phi_{u-1}(\mathbf{y}_t(\mathbf{z} + \mathbf{l}_t)) - \phi_u(\mathbf{y}_t(\mathbf{z} + \mathbf{l}_t))),$$

can be written as,

$$\zeta_u(\mathbf{y}_t(\mathbf{l}_t)) = \frac{1}{\kappa} \sum_{\mathbf{z}' \in \{\Omega - \mathbf{l}_t\}} K(\mathbf{z}' - \mathbf{l}_t) \cdot (\phi_{u-1}(\mathbf{y}_t(\mathbf{z}')) - \phi_u(\mathbf{y}_t(\mathbf{z}'))), \quad (33)$$

Following the formulation in Hager et al. (2004), we define the *sifting* vector,

$$\tilde{\mathbf{u}}_j = \begin{bmatrix} \phi_{j-1}(\mathbf{y}_t(\mathbf{z}'_1)) - \phi_j(\mathbf{y}_t(\mathbf{z}'_1)) \\ \phi_{j-1}(\mathbf{y}_t(\mathbf{z}'_2)) - \phi_j(\mathbf{y}_t(\mathbf{z}'_2)) \\ \vdots \\ \phi_{j-1}(\mathbf{y}_t(\mathbf{z}'_N)) - \phi_j(\mathbf{y}_t(\mathbf{z}'_N)) \end{bmatrix}. \quad (34)$$

We can then combine these sifting vectors into the *sifting matrix*, $\tilde{\mathbf{U}} = [\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_B]$. Similarly, we can define the kernel vector, $\mathbf{K}(\mathbf{z}) = \frac{1}{\kappa} [K(\mathbf{z}_1), K(\mathbf{z}_2), \dots, K(\mathbf{z}_N)]^\top$, where $N = |\Omega|$ and the indexing of the pixels is performed column wise. Therefore we can write the full kernel-weighted histogram, ζ as,

$$\zeta(\mathbf{y}_t(\mathbf{l}_t)) = \tilde{\mathbf{U}}^\top \mathbf{K}(\mathbf{z}' - \mathbf{l}_t) \quad (35)$$

Since $\tilde{\mathbf{U}}$ is not a function of \mathbf{l}_t ,

$$\nabla_{\mathbf{l}_t}(\zeta(\mathbf{y}_t(\mathbf{l}_t))) = \tilde{\mathbf{U}}^\top \mathbf{J}_K, \quad (36)$$

where,

$$\mathbf{J}_K = \frac{1}{\kappa} \left[\frac{\partial \mathbf{K}(\mathbf{z}' - \mathbf{l}_t)}{\partial \mathbf{l}_{t,1}}, \frac{\partial \mathbf{K}(\mathbf{z}' - \mathbf{l}_t)}{\partial \mathbf{l}_{t,2}} \right] = [\nabla K(\mathbf{z}'_1 - \mathbf{l}_t), \nabla K(\mathbf{z}'_2 - \mathbf{l}_t), \dots, \nabla K(\mathbf{z}'_N - \mathbf{l}_t)]^\top. \quad (37)$$

where $\nabla K(\mathbf{z}' - \mathbf{l}_t)$ is the derivative of the kernel function, e.g., the Epanechnikov kernel in Eq. (10). Therefore the derivative of the first kernel-weighted histogram, $\sqrt{\zeta(\mathbf{y}_t(\mathbf{l}_t))}$ w.r.t. \mathbf{l}_t is,

$$\mathbf{L} \doteq \frac{1}{2} \text{diag}(\zeta(\mathbf{y}_t(\mathbf{l}_t)))^{-\frac{1}{2}} \tilde{\mathbf{U}}^\top \mathbf{J}_K \quad (38)$$

where $\text{diag}(\mathbf{v})$ creates a diagonal matrix with \mathbf{v} on its diagonal and 0 in its off-diagonal entries. Since $\mathbf{y}_t(\mathbf{l}_t)$ does not depend on the state of the dynamic template, \mathbf{x}_t , the derivative of the first kernel-weighted histogram w.r.t. \mathbf{x}_t is 0.

In the same manner, the expression for the second kernel weighted histogram, for bin u ,

$$\zeta_u(\mu + C\mathbf{x}_t) = \frac{1}{\kappa} \sum_{\mathbf{z} \in \Omega} K(\mathbf{z}) \cdot \left(\phi_{u-1}(\mu(\mathbf{z}) + C(\mathbf{z})^\top \mathbf{x}_t) - \phi_u(\mu(\mathbf{z}) + C(\mathbf{z})^\top \mathbf{x}_t) \right) \quad (39)$$

By using a similar sifting vector for the predicted dynamic template,

$$\Phi_j = \begin{bmatrix} (\phi_{j-1} - \phi_j)(\mu(\mathbf{z}_1) + C(\mathbf{z}_1)^\top \mathbf{x}_t) \\ (\phi_{j-1} - \phi_j)(\mu(\mathbf{z}_2) + C(\mathbf{z}_2)^\top \mathbf{x}_t) \\ \vdots \\ (\phi_{j-1} - \phi_j)(\mu(\mathbf{z}_N) + C(\mathbf{z}_N)^\top \mathbf{x}_t) \end{bmatrix}, \quad (40)$$

where for brevity, we use

$$(\phi_{j-1} - \phi_j)(\mu(\mathbf{z}) + C(\mathbf{z})^\top \mathbf{x}_t) = \phi_{j-1}(\mu(\mathbf{z}) + C(\mathbf{z})^\top \mathbf{x}_t) - \phi_j(\mu(\mathbf{z}) + C(\mathbf{z})^\top \mathbf{x}_t).$$

and the pixel indices \mathbf{z}_j are used in a column-wise fashion as discussed in the main text. Using the corresponding sifting matrix, $\Phi = [\Phi_1, \Phi_2, \dots, \Phi_B] \in \mathbb{R}^{N \times B}$, we can write,

$$\zeta(\mu + C\mathbf{x}_t) = \Phi^\top \text{diag}(\mathbf{K}(\mathbf{z})) \quad (41)$$

Since $\zeta(\mu + C\mathbf{x}_t)$ is only a function of \mathbf{x}_t ,

$$\nabla_{\mathbf{x}_t}(\zeta(\mu + C\mathbf{x}_t)) = (\Phi')^\top \text{diag}(\mathbf{K}(\mathbf{z}))C, \quad (42)$$

where $\Phi' = [\Phi'_1, \Phi'_2, \dots, \Phi'_B]$, is the derivative of the sifting matrix with,

$$\Phi'_j = \begin{bmatrix} (\phi'_{j-1} - \phi'_j)(\mu(\mathbf{z}_1) + C(\mathbf{z}_1)^\top \mathbf{x}_t) \\ (\phi'_{j-1} - \phi'_j)(\mu(\mathbf{z}_2) + C(\mathbf{z}_2)^\top \mathbf{x}_t) \\ \vdots \\ (\phi'_{j-1} - \phi'_j)(\mu(\mathbf{z}_N) + C(\mathbf{z}_N)^\top \mathbf{x}_t) \end{bmatrix}, \quad (43)$$

and Φ' is the derivative of the sigmoid function. Therefore, the derivative of the second kernel-weighted histogram, $\sqrt{\zeta(\mu + C\mathbf{x}_t)}$ w.r.t. \mathbf{x}_t is,

$$\mathbf{M} \doteq \frac{1}{2} \text{diag}(\zeta(\mu + C\mathbf{x}_t))^{-\frac{1}{2}} (\Phi')^\top \text{diag}(\mathbf{K}(\mathbf{z}))C. \quad (44)$$

The second part of the cost function in Eq. (32) depends only on the current state, and the derivative w.r.t. \mathbf{x}_t can be simply computed as,

$$\mathbf{d} \doteq Q^{-1}(\mathbf{x}_t - A\hat{\mathbf{x}}_{t-1}). \quad (45)$$

When computing the derivatives of the squared difference function in first term in Eq. (32), the difference,

$$\mathbf{a} \doteq \sqrt{\zeta(\mathbf{y}_t(\mathbf{l}_t))} - \sqrt{\zeta(\mu + C\mathbf{x}_t)}, \quad (46)$$

will be multiplied with the derivatives of the individual square root kernel-weighted histograms.

Finally, the derivative of the cost function in Eq. (32) w.r.t. \mathbf{l}_t is computed as,

$$\nabla_{\mathbf{l}_t} O(\mathbf{l}_t, \mathbf{x}_t) = \frac{1}{\sigma_H^2} \mathbf{L}^\top \mathbf{a}, \quad (47)$$

and the derivative of the cost function Eq. (32) w.r.t. \mathbf{x}_t is computed as,

$$\nabla_{\mathbf{x}_t} O(\mathbf{l}_t, \mathbf{x}_t) = \frac{1}{\sigma_H^2} (-\mathbf{M})^\top \mathbf{a} + \mathbf{d}. \quad (48)$$

We can incorporate the σ_H^{-2} term in \mathbf{L} and \mathbf{M} to get the gradient descent optimization scheme in Eq. (26).

References

- Ali S, Shah M (2010) Human action recognition in videos using kinematic features and multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2):288–303
- Babenko B, Yang MH, Belongie S (2009) Visual tracking with online multiple instance learning. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Bissacco A, Chiuso A, Ma Y, Soatto S (2001) Recognition of human gaits. In: *IEEE Conference on Computer Vision and Pattern Recognition*, vol 2, pp 52–58

- Bissacco A, Chiuso A, Soatto S (2007) Classification and recognition of dynamical models: The role of phase, independent components, kernels and optimal transport. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(11):1958–1972
- Black MJ, Jepson AD (1998) Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision* 26(1):63–84
- Chan A, Vasconcelos N (2007) Classifying video with kernel dynamic textures. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp 1–6
- Chaudhry R, Vidal R (2009) Recognition of visual dynamical processes: Theory, kernels and experimental evaluation. Tech. Rep. 09-01, Department of Computer Science, Johns Hopkins University
- Chaudhry R, Ravichandran A, Hager G, Vidal R (2009) Histograms of oriented optical flow and Binet-Cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Cock KD, Moor BD (2002) Subspace angles and distances between ARMA models. *System and Control Letters* 46(4):265–270
- Collins R, Liu Y, Leordeanu M (2005a) On-line selection of discriminative tracking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*
- Collins R, Zhou X, Teh SK (2005b) An open source tracking testbed and evaluation web site. In: *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2005)*
- Comaniciu D, Meer P (2002) Mean Shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(5):603–619
- Comaniciu D, Ramesh V, Meer P (2003) Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25(5):564–577
- Doretto G, Chiuso A, Wu Y, Soatto S (2003) Dynamic textures. *International Journal of Computer Vision* 51(2):91–109
- Efros A, Berg A, Mori G, Malik J (2003) Recognizing action at a distance. In: *IEEE Int. Conf. on Computer Vision*, pp 726–733
- Fan Z, Yang M, Wu Y (2007) Multiple collaborative kernel tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29:1268–1273
- Felzenszwalb P, Girshick R, McAllester D, Ramanan D (2010) Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32
- Gill PE, Murray W, Wright MH (1987) *Practical Optimization*. Academic Press
- Gorelick L, Blank M, Shechtman E, Irani M, Basri R (2007) Actions as space-time shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(12):2247–2253
- Grabner H, Grabner M, Bischof H (2006) Real-time tracking via on-line boosting. In: *British Machine Vision Conference*
- Hager GD, Dewan M, Stewart CV (2004) Multiple kernel tracking with ssd. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Ikizler N, Duygulu P (2009) Histogram of oriented rectangles: A new pose descriptor for human action recognition. *Image and Vision Computing* 27 (10):1515–1526
- Isard M, Blake A (1998) Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision* 29(1):5–28
- Jepson AD, Fleet DJ, El-Maraghi TF (2001) Robust online appearance models for visual tracking. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Leibe B, Schindler K, Cornelis N, Gool LV (2008) Coupled object detection and tracking from static cameras and moving vehicles. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30(10):1683–1698
- Lim H, Morariu VI, Camps OI, Sznaiar M (2006) Dynamic appearance modeling for human tracking. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Lin Z, Jiang Z, Davis LS (2009) Recognizing actions by shape-motion prototype trees. In: *IEEE Int. Conf. on Computer Vision*
- Nejhum SMS, Ho J, Yang MH (2008) Visual tracking with histograms and articulating blocks. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Niebles JC, Wang H, Fei-Fei L (2008) Unsupervised learning of human action categories using spatial-temporal words. *International Journal of Computer Vision* 79:299–318
- North B, Blake A, Isard M, Rittscher J (2000) Learning and classification of complex dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(9):1016–1034
- Pavlovic V, Rehg JM, Cham TJ, Murphy KP (1999) A dynamic Bayesian network approach to figure tracking using learned dynamic models. In: *IEEE Int. Conf. on Computer Vision*, pp 94–101
- Péteri R (2010) Tracking dynamic textures using a particle filter driven by intrinsic motion information. *Vision and Applications, special Issue on Dynamic Textures in Video*
- Péteri R, Fazekas S, Huskies M (2010) DynTex: A comprehensive database of dynamic textures. *Pattern Recognition Letters* 31:1627–1632, URL www.cwi.nl/projects/dyntex/, online Dynamic Texture Database
- Ravichandran A, Vidal R (2008) Video registration using dynamic textures. In: *European Conference on Computer Vision*
- Ravichandran A, Vidal R (2011) Dynamic texture registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33:158–171
- Ravichandran A, Chaudhry R, Vidal R (2009) View-invariant dynamic texture recognition using a bag of dynamical systems. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Saisan P, Doretto G, Wu YN, Soatto S (2001) Dynamic texture recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*, vol II, pp 58–63
- Thurau C, Hlavac V (2008) Pose primitive based human action recognition in videos or still images. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Vidal R, Ravichandran A (2005) Optical flow estimation and segmentation of multiple moving dynamic textures. In: *IEEE Conference on Computer Vision and Pattern Recognition*, vol II, pp 516–521
- Xie Y, Chang H, Li Z, Liang L, Chen X, Zhao D (2011) A unified framework for locating and recognizing human actions. In: *IEEE Conference on Computer Vision and Pattern Recognition*
- Yilmaz A, Javed O, Shah M (2006) Object tracking: A survey. *ACM Computing Surveys* 38(4):13